

SQL machine learning documentation

Learn how to use machine learning on SQL to run Python and R on relational data, both on-premises and in the cloud. Tutorials, code examples, installation guides, and other documentation show you how to use SQL machine learning.

Machine learning on SQL Server

OVERVIEW

[What is SQL Server Machine Learning Services with Python and R?](#)

GET STARTED

[Install Machine Learning Services on Windows](#)

[Install Machine Learning Services on Linux](#)

VIDEO

[How to Execute R/Python in SQL Server with Machine Learning Services](#) 

Get started with Python

QUICKSTART

[Run simple Python scripts](#)

[Handle inputs and outputs](#)

[Use Python data structures in SQL](#)

[Train and use your first model](#)

TUTORIAL

[Predict ski rental with linear regression](#)

[Categorizing customers using k-means clustering](#)

Get started with R

QUICKSTART

[Run simple R scripts](#)

[Handle inputs and outputs](#)

[Use R data structures in SQL](#)

[Train and use your first model](#)

TUTORIAL

[Predict ski rental with decision tree](#)

[Categorizing customers using k-means clustering](#)

Azure SQL Managed Instance

OVERVIEW

[Machine Learning Services in Azure SQL Managed Instance](#)

[Key differences between ML in Azure SQL Managed Instance and SQL Server](#)

HOW-TO GUIDE

[Deploy and make predictions with an ONNX model in Azure SQL Managed Instance](#)

Azure SQL Edge

OVERVIEW

[Machine learning and AI with ONNX in Azure SQL Edge](#)

HOW-TO GUIDE

[Deploy and make predictions with an ONNX model in Azure SQL Edge](#)

Azure Data Studio ML extension

OVERVIEW

What is the Machine Learning extension for Azure Data Studio?

 **HOW-TO GUIDE**

[Manage packages in database](#)

[Make predictions](#)

[Import or view models](#)

SQL Server Big Data Clusters

 **OVERVIEW**

[What are SQL Server Big Data Clusters?](#)

 **GET STARTED**

[Enable Machine Learning Services on SQL Server Big Data Clusters](#)

 **HOW-TO GUIDE**

[Spark machine learning models with MLeap](#)

[Sparklyr in SQL Server Big Data Cluster](#)

[Integrate Spark with SQL Server](#)

Azure Synapse Analytics

 **OVERVIEW**

[Machine Learning capabilities in Azure Synapse Analytics](#)

[Train machine learning models](#)

 **QUICKSTART**

[Create a new Azure Machine Learning linked service in Synapse](#)

 **TUTORIAL**

[Train a model in Python with automated machine learning](#)

Build a machine learning app with Apache Spark MLlib and Azure Synapse Analytics


Microsoft SQL documentation

Learn how to use SQL Server and Azure SQL, both on-premises and in the cloud.



DOWNLOAD
[Try SQL Server 2022](#) 



OVERVIEW
[Get an Azure VM with SQL Server](#) 



OVERVIEW
[SQL tools](#)



REFERENCE
[Troubleshoot](#)



SQL Server on Windows

[On-premises](#)

[On an Azure VM](#)

[Previous Versions \(SQL 2005-2012\)](#)



SQL on Azure

[Azure SQL Database](#)

[Azure SQL Managed Instance](#)

[Azure SQL Edge](#)

[Azure Synapse Analytics](#)









SQL Server on Linux







[On-premises](#)

[On an Azure VM](#)

Getting started

-  [Connect & query \(Azure Data Studio\)](#)
-  [Connect & query \(SSMS\)](#)
-  [Writing Transact-SQL](#)
-  [SQL Server in a Linux container](#)
-  [SQL Docs navigation tips](#)
-  [Educational SQL resources](#)

What's new

-  [SQL Server 2022](#)
-  [Azure SQL Database](#)
-  [Azure SQL Managed Instance](#)
-  [SQL Server on Azure VMs](#)
-  [Azure Data Studio](#)
-  [SQL Server Machine Learning Services](#)

 [Azure Arc-enabled SQL Server](#)

Install

-  [SQL Server](#)
-  [SQL Server Management Studio \(SSMS\)](#)
-  [Azure Data Studio](#)
-  [Analysis Services \(SSAS\)](#)
-  [Integration Services \(SSIS\)](#)
-  [Reporting Services \(SSRS\)](#)

Migrate

-  [Data Migration Guide](#)
-  [Database Compatibility Certification](#)
-  [Azure Database Migration Service \(DMS\)](#)
-  [Database Migration Assistant \(DMA\)](#)
-  [Database Experimentation Assistant \(DEA\)](#)
-  [SQL Server Migration Assistant \(SSMA\)](#)

[See more >](#)

Development

- [Build an app using SQL Server !\[\]\(c6a8736a601a632e2c96605cf66055ed_img.jpg\)](#)
- [SQL Server client connectivity](#)
- [DevOps using SQL Server !\[\]\(64ef2b19d70b31fbbfce0e0e2aa3d7b4_img.jpg\)](#)
- [Azure SQL Database applications](#)
- [Azure SQL Database connectivity libraries](#)
- [Transact-SQL \(T-SQL\)](#)
- [SQL PowerShell](#)
- [Database samples](#)

Reporting & analysis

- [Power BI Report Server](#)
- [Power BI Service](#)
- [SQL Server Reporting Services \(SSRS\)](#)
- [SQL Server Analysis Services \(SSAS\)](#)
- [Azure Analysis Services](#)

Big data & machine learning

- [Azure Synapse Analytics](#)
- [SQL Server Machine Learning Services \(R & Python\)](#)
- [SQL Server Big Data Clusters](#)
- [Data Virtualization with PolyBase](#)
- [Microsoft Analytics Platform System](#)

What is SQL Server Machine Learning Services with Python and R?

Article • 08/01/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

Machine Learning Services is a feature in SQL Server that gives the ability to run Python and R scripts with relational data. You can use open-source packages and frameworks, and the [Microsoft Python and R packages](#), for predictive analytics and machine learning. The scripts are executed in-database without moving data outside SQL Server or over the network. This article explains the basics of SQL Server Machine Learning Services and how to get started.

Note

Machine Learning Services is also available in [Azure SQL Managed Instance](#). For machine learning on other SQL platforms, see the [SQL machine learning documentation](#).

Execute Python and R scripts in SQL Server

SQL Server Machine Learning Services lets you execute Python and R scripts in-database. You can use it to prepare and clean data, do feature engineering, and train, evaluate, and deploy machine learning models within a database. The feature runs your scripts where the data resides and eliminates transfer of the data across the network to another server.

You can execute Python and R scripts on a SQL Server instance with the stored procedure [sp_execute_external_script](#).

Base distributions of Python and R are included in Machine Learning Services. You can install and use open-source packages and frameworks, such as PyTorch, TensorFlow, and scikit-learn, in addition to the Microsoft packages.

Machine Learning Services uses an extensibility framework to run Python and R scripts in SQL Server. Learn more about how this works:

- [Extensibility framework](#)
- [Python extension](#)
- [R extension](#)

Get started with Machine Learning Services

1. [Install SQL Server Machine Learning Services on Windows.](#)
2. Configure your development tools. You can use [run Python and R scripts in Azure Data Studio notebooks](#). You can also use T-SQL in [Azure Data Studio](#).
3. Write your first Python or R script.
 - [Python tutorials for SQL machine learning](#)
 - [R tutorials for SQL machine learning](#)

Python and R versions

The following lists the versions of Python and R that are included in Machine Learning Services.

SQL Server version	Cumulative Update	Python runtime version	R runtime versions
SQL Server 2022*	RTM and later	3.10.2	4.2.0
SQL Server 2019	RTM and later	3.7.1	3.5.2
SQL Server 2017	CU22 and later	3.5.2 and 3.7.2	3.3.3 and 3.5.2
SQL Server 2017	RTM - CU21	3.5.2	3.3.3
SQL Server 2016	See the R version		

* For supported versions of R and Python and the RevoScaleR and revoscalepy packages, see [Install SQL Server 2022 Machine Learning Services \(Python and R\) on Windows](#) or [Install SQL Server Machine Learning Services \(Python and R\) on Linux](#).

Python and R packages

You can use open-source packages and frameworks, in addition to Microsoft's enterprise packages. Most common open-source Python and R packages are pre-installed in Machine Learning Services.

ⓘ Note

Beginning with SQL Server 2022 (16.x), runtimes for R, Python, and Java, are no longer installed with SQL Setup. Instead, install your desired R and/or Python custom runtime(s) and packages. For more information, see [Install SQL Server](#)

2022 Machine Learning Services on Windows or Install SQL Server Machine Learning Services (Python and R) on Linux.

The following Python and R packages from Microsoft are also included at installation:

Language	Package	Description
Python	revoscalepy	The primary package for scalable Python. Data transformations and manipulation, statistical summarization, visualization, and many forms of modeling. Additionally, functions in this package automatically distribute workloads across available cores for parallel processing.
Python	microsoftml	Applies only to SQL Server 2016, SQL Server 2017, and SQL Server 2019. Adds machine learning algorithms to create custom models for text analysis, image analysis, and sentiment analysis.
R	RevoScaleR	The primary package for scalable R. Data transformations and manipulation, statistical summarization, visualization, and many forms of modeling. Additionally, functions in this package automatically distribute workloads across available cores for parallel processing.
R	MicrosoftML (R)	Applies only to SQL Server 2016, SQL Server 2017, and SQL Server 2019. Adds machine learning algorithms to create custom models for text analysis, image analysis, and sentiment analysis.
R	olapR	Applies only to SQL Server 2016, SQL Server 2017, and SQL Server 2019. R functions used for MDX queries against a SQL Server Analysis Services OLAP cube.
R	sqlrutils	Applies only to SQL Server 2016, SQL Server 2017, and SQL Server 2019. A mechanism to use R scripts in a T-SQL stored procedure, register that stored procedure with a database, and run the stored procedure from an R development environment .
R	Microsoft R Open (retired ↗)	Applies to: SQL Server 2016, SQL Server 2017, and SQL Server 2019. Microsoft R Open (MRO) was the enhanced distribution of R from Microsoft.

For more information on which packages are installed with Machine Learning Services and how to install other packages, see:

- [Get Python package information](#)
- [Install packages with Python tools on SQL Server](#)
- [Get R package information](#)

- [Use T-SQL \(CREATE EXTERNAL LIBRARY\) to install R packages on SQL Server.](#)

Next steps

- [Install SQL Server Machine Learning Services on Windows or on Linux](#)
- [Python tutorials for SQL machine learning](#)
- [R tutorials for SQL machine learning](#)

What are standalone Machine Learning Server or R Server in SQL Server?

Article • 08/01/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Important

The support for Machine Learning Server (previously known as R Server) ended on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

SQL Server provides installation support for a standalone R Server or Machine Learning Server that runs independently of SQL Server. Depending on your SQL Server version, a standalone server has a foundation of open-source R and possibly Python, overlaid with high-performance libraries from Microsoft that add statistical and predictive analytics at scale. Libraries also enable machine learning tasks scripted in R or Python.

In SQL Server 2016, this feature is called **R Server (Standalone)** and is R-only. In SQL Server 2017, it's called **Machine Learning Server (Standalone)** and includes both R and Python.

Note

As installed by SQL Server Setup, a standalone server is functionally equivalent to the non-SQL-branded versions of **Microsoft Machine Learning Server**, supporting the same user scenarios, including remote execution, operationalization and web services, and the complete collection of R and Python libraries.

Components

SQL Server 2016 is R only. SQL Server 2017 supports R and Python. The following table describes the features in each version.

Component	Description
R packages	RevoScaleR is the primary library for scalable R with functions for data manipulation, transformation, visualization, and analysis. MicrosoftML adds machine learning algorithms to create custom models for text

Component	Description
	analysis, image analysis, and sentiment analysis. sqlRUtils provides helper functions for putting R scripts into a T-SQL stored procedure, registering a stored procedure with a database, and running the stored procedure from an R development environment. olapR is for specifying MDX queries in R.
Microsoft R Open (MRO)	Microsoft R Open (retired) was Microsoft's open-source distribution of R.
R tools	R console windows and command prompts are standard tools in an R distribution. Find them at <code>\Program files\Microsoft SQL Server\140\R_SERVER\bin\x64</code> .
R Samples and scripts	Open-source R and RevoScaleR packages include built-in data sets so that you can create and run script using pre-installed data. Look for them at <code>\Program files\Microsoft SQL Server\140\R_SERVER\library\datasets</code> and <code>\library\RevoScaleR</code> .
Python packages	revoscalepy is the primary library for scalable Python with functions for data manipulation, transformation, visualization, and analysis. microsoftml adds machine learning algorithms to create custom models for text analysis, image analysis, and sentiment analysis.
Python tools	The built-in Python command-line tool is useful for ad hoc testing and tasks. Find the tool at <code>\Program files\Microsoft SQL Server\140\PYTHON_SERVER\python.exe</code> .
Anaconda	Anaconda is an open-source distribution of Python and essential packages.
Python samples and scripts	As with R, Python includes built-in data sets and scripts. Find the <code>revoscalepy</code> data at <code>\Program files\Microsoft SQL Server\140\PYTHON_SERVER\lib\site-packages\revoscalepy\data\sample-data</code> .
Pre-trained models in R and Python	Pre-trained models are created for specific use cases and maintained by the data science engineering team at Microsoft. You can use the pre-trained models as-is to score positive-negative sentiment in text, or detect features in images, using new data inputs that you provide. Pre-trained models are supported and usable on a standalone server, but you cannot install them through SQL Server Setup. For more information, see Install pretrained machine learning models on SQL Server .

Using a standalone server

R and Python developers typically choose a standalone server to move beyond the memory and processing constraints of open-source R and Python. R and Python libraries executing on a standalone server can load and process large amounts of data on multiple cores and aggregate the results into a single consolidated output. High-

performance functions are engineered for both scale and utility: delivering predictive analytics, statistical modeling, data visualizations, and leading-edge machine learning algorithms in a commercial server product engineered and supported by Microsoft.

As an independent server decoupled from SQL Server, the R and Python environment is configured, secured, and accessed using the underlying operating system and standard tools provided in the standalone server, not SQL Server. There is no built-in support for SQL Server relational data. If you want to use SQL Server data, you can create data source objects and connections as you would from any client.

As an adjunct to SQL Server, a standalone server is also useful as a powerful development environment if you need both local and remote computing. The R and Python packages on a standalone server are the same as those provided with a database engine installation, allowing for code portability and [compute-context switching](#).

How to get started

Start with setup, attach the binaries to your favorite development tool, and write your first script.

Step 1: Install the software

Install either one of these versions:

- [SQL Server 2017 Machine Learning Server \(standalone\)](#)
- [SQL Server 2016 R Server \(Standalone\) - R only](#)

Step 2: Configure a development tool

On a standalone server, it's common to work locally using a development installed on the same computer.

- [Set up R tools](#)
- [Set up Python tools](#)

Step 3: Write your first script

Write R or Python script using functions from RevoScaleR, revoscalepy, and the machine learning algorithms.

- [Explore R and RevoScaleR in 25 Functions](#): Start with basic R commands and then progress to the RevoScaleR distributable analytical functions that provide high

performance and scaling to R solutions. Includes parallelizable versions of many of the most popular R modeling packages, such as k-means clustering, decision trees, and decision forests, and tools for data manipulation.

- [Quickstart: An example of binary classification with the microsoftml Python package](#): Create a binary classification model using the functions from microsoftml and the well-known breast cancer dataset.

Choose the best language for the task. R is best for statistical computations that are difficult to implement using SQL. For set-based operations over data, leverage the power of SQL Server to achieve maximum performance. Use the in-memory database engine for very fast computations over columns.

Step 4: Operationalize your solution

Standalone servers can use the [operationalization](#) functionality of the non-SQL-branded [Microsoft Machine Learning Server](#). You can configure a standalone server for operationalization, which gives you these benefits: deploy and host your code as web services, run diagnostics, test web service capacity.

Step 5: Maintain your server

SQL Server releases cumulative updates on a regular basis. Applying the cumulative updates adds security and functional enhancements to an existing installation.

Descriptions of new or changed functionality can be found in the [CAB Downloads](#) article and on the web pages for [SQL Server 2016 cumulative updates](#) [↗](#) and [SQL Server 2017 cumulative updates](#) [↗](#).

For more information on how to apply updates to an existing instance, see [Apply updates](#) in the installation instructions.

See also

[Install R Server \(Standalone\) or Machine Learning Server \(Standalone\)](#)

What's new in SQL Server Machine Learning Services?

Article • 03/03/2023


Applies to:  SQL Server 2016 (13.x) and later versions

This article describes what new capabilities and features are included in each version of [SQL Server Machine Learning Services](#). Machine learning capabilities are added to SQL Server in each release as we continue to expand, extend, and deepen the integration between the data platform, advanced analytics, and data science.

Note

Feature capabilities and installation options vary between versions of SQL Server. Use the version selector dropdown to choose the appropriate version of SQL Server.

New in SQL Server 2017

This release adds [Python support and industry-leading machine learning algorithms](#) . Renamed to reflect the new scope, SQL Server 2017 marks the introduction of [SQL Server Machine Learning Services \(In-Database\)](#), with language support for both Python and R.

For feature announcements all-up, see [What's New in SQL Server 2017](#).

R enhancements

The R component of SQL Server Machine Learning Services is the next generation of SQL Server 2016 R Services, with updated versions of base R, RevoScaler, and other packages.

New capabilities for R include [package management](#), with the following highlights:

- Database roles help DBAs manage packages and assign permissions for package installation.
- [CREATE EXTERNAL LIBRARY](#) helps DBAs manage packages in the familiar T-SQL language.

- [RevoScaleR](#) functions help install, remove, or list packages owned by users. For more information, see [How to use RevoScaleR functions to find or install R packages on SQL Server](#).

R libraries

Package	Description
MicrosoftML	In this release, MicrosoftML is included in a default R installation, eliminating the upgrade step required in the previous SQL Server 2016 R Services. MicrosoftML provides state-of-the-art machine learning algorithms and data transformations that can be scaled or run in remote compute contexts. Algorithms include customizable deep neural networks, fast decision trees and decision forests, linear regression, and logistic regression.

Python integration for in-database analytics

Python is a language that offers great flexibility and power for a variety of machine learning tasks. Open-source libraries for Python include several platforms for customizable neural networks, as well as popular libraries for natural language processing.

Because Python is integrated with the database engine, you can keep analytics close to the data and eliminate the costs and security risks associated with data movement. You can deploy machine learning solutions based on Python using tools like Visual Studio. Your production applications can get predictions, models, or visuals from the Python 3.5 runtime using SQL Server data access methods.

T-SQL and Python integration is supported through the [sp_execute_external_script](#) system stored procedure. You can call any Python code using this stored procedure. Code runs in a secure, dual architecture that enables enterprise-grade deployment of Python models and scripts, callable from an application using a simple stored procedure. Additional performance gains are achieved by streaming data from SQL to Python processes and MPI ring parallelization.

You can use the T-SQL [PREDICT](#) function to perform [native scoring](#) on a pre-trained model that has been previously saved in the required binary format.

Python libraries

Package	Description
---------	-------------

Package	Description
revoscalepy	Python-equivalent of RevoScaleR. You can create Python models for linear and logistic regressions, decision trees, boosted trees, and random forests, all parallelizable, and capable of being run in remote compute contexts. This package supports use of multiple data sources and remote compute contexts. The data scientist or developer can execute Python code on a remote SQL Server, to explore data or build models without moving data.
microsoftml	Python-equivalent of the MicrosoftML R package.

Pre-trained models

[Pre-trained models](#) are available for both Python and R. Use these models for image recognition and positive-negative sentiment analysis, to generate predictions on your own data.

Standalone Server as a shared feature in SQL Server Setup

This release also adds [SQL Server Machine Learning Server \(Standalone\)](#), a fully independent data science server, supporting statistical and predictive analytics in R and Python. As with R Services, this server is the next version of SQL Server 2016 R Server (Standalone). With the standalone server, you can distribute and scale R or Python solutions with no dependencies on SQL Server.

New in SQL Server 2016

This release introduced machine learning capabilities into SQL Server through [SQL Server 2016 R Services](#), an in-database analytics engine for processing R script on resident data within a database engine instance.

Additionally, [SQL Server 2016 R Server \(Standalone\)](#) was released as a way to install R Server on a Windows server. Initially, SQL Server Setup provided the only way to install R Server for Windows. In later releases, developers and data scientists who wanted R Server on Windows could use another standalone installer to achieve the same goal. The standalone server in SQL Server is functionally equivalent to the standalone server product, [Microsoft R Server for Windows](#).

For feature announcements all-up, see [What's New in SQL Server 2016](#).

Release	Feature update
---------	----------------

Release	Feature update
CU additions	<p>Real-time scoring relies on native C++ libraries to read a model stored in an optimized binary format, and then generate predictions without having to call the R runtime. This makes scoring operations much faster. With real-time scoring, you can run a stored procedure or perform real-time scoring from R code. Real-time scoring is also available for SQL Server 2016, if the instance is upgraded to the latest release of Microsoft R Server.</p>
Initial release	<p>R integration for in-database analytics.</p> <p>R packages for calling R functions in T-SQL, and vice versa. RevoScaleR functions provide R analytics at scale by chunking data into component parts, coordinating and managing distributed processing, and aggregating results. In SQL Server 2016 R Services (In-Database), the RevoScaleR engine is integrated with a database engine instance, bringing data and analytics together in the same processing context.</p> <p>T-SQL and R integration through <code>sp_execute_external_script</code>. You can call any R code using this stored procedure. This secure infrastructure enables enterprise-grade deployment of Rn models and scripts that can be called from an application using a simple stored procedure. Additional performance gains are achieved by streaming data from SQL to R processes and MPI ring parallelization.</p> <p>You can use the T-SQL <code>PREDICT</code> function to perform native scoring on a pre-trained model that has been previously saved in the required binary format.</p>

Linux support

SQL Server 2019 adds Linux support for R and Python when you install the machine learning packages with a database engine instance. For more information, see [Install SQL Server Machine Learning Services on Linux](#).

On Linux, SQL Server 2017 does not have R or Python integration, but you can use **Native scoring** on Linux because that functionality is available through T-SQL `PREDICT`, which runs on Linux. Native scoring enables high-performance scoring from a pretrained model, without calling or even requiring an R runtime.

Next steps

- [Install SQL Server Machine Learning Services \(In-Database\)](#)

Install SQL Server Machine Learning Services (Python and R) on Windows

Article • 03/17/2023

Applies to: ✔ SQL Server 2016 (13.x), ✔ SQL Server 2017 (14.x), and ✔ SQL Server 2019 (15.x)

This article shows you how to install [SQL Server Machine Learning Services](#) on Windows. You can use Machine Learning Services to run Python and R scripts in-database.

📘 Important

These instructions apply to SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x). For SQL Server 2022 (16.x), refer to [Install SQL Server 2022 Machine Learning Services on Windows](#).

Pre-installation checklist

- A database engine instance is required. You can't install just Python or R features, although you can add them incrementally to an existing instance.
- For business continuity, [Always On availability groups](#) are supported for Machine Learning Services. Install Machine Learning Services, and configure packages, on each node.
- Installing Machine Learning Services *is not supported* on an [Always On failover cluster instance](#) in SQL Server 2017. It's supported with SQL Server 2019 and later.
- Don't install Machine Learning Services on a domain controller. The Machine Learning Services portion of setup will fail.
- Don't install **Shared Features > Machine Learning Server (Standalone)** on the same computer that's running a database instance. A standalone server will compete for the same resources and diminish the performance of both installations.
- Side-by-side installation with other versions of Python and R is supported, but we don't recommend it. It's supported because the SQL Server instance uses its own copies of the open-source R and Anaconda distributions. We don't recommend it

because running code that uses Python and R on a computer outside SQL Server can lead to problems:

- Using a different library and different executable files will create results that are inconsistent with what you're running in SQL Server.
- SQL Server can't manage R and Python scripts that run in external libraries, leading to resource contention.

Important

After you finish setup, be sure to complete the post-configuration steps described in this article. These steps include enabling SQL Server to use external scripts and adding accounts that are required for SQL Server to run R and Python jobs on your behalf. Configuration changes generally require a restart of the instance or a restart of the Launchpad service.

Get the installation media

The download location for SQL Server depends on the edition:

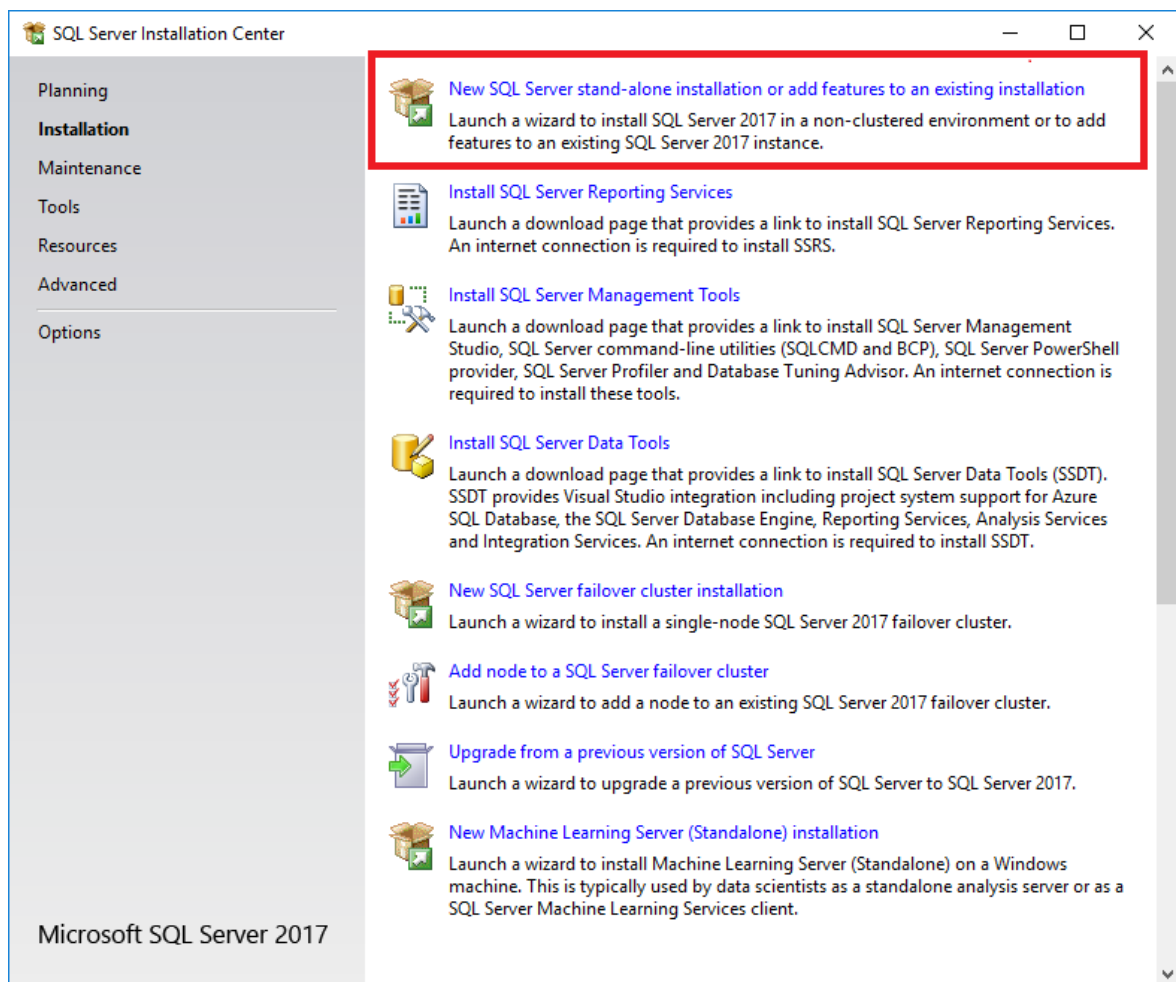
- **SQL Server Enterprise, Standard, and Express editions.** These editions are licensed for production use. For the Enterprise and Standard editions, contact your software vendor for the installation media. You can find purchasing information and a directory of Microsoft partners on the [Microsoft purchasing website](#).
- [The latest free edition](#).

For more information on which SQL Server editions support Python and R integration with Machine Learning Services, see [Editions and supported features of SQL Server 2017](#).

Run setup

For local installations, you must run the setup as an administrator. If you install SQL Server from a remote share, you must use a domain account that has read and execute permissions on the remote share.

1. Start the setup wizard for SQL Server.
2. On the **Installation** tab, select **New SQL Server stand-alone installation or add features to an existing installation**.



3. On the **Feature Selection** page, select these options:

- **Database Engine Services**

To use R and Python with SQL Server, you must install an instance of the database engine. You can use either a default instance or a named instance.

- **Machine Learning Services (In-Database)**

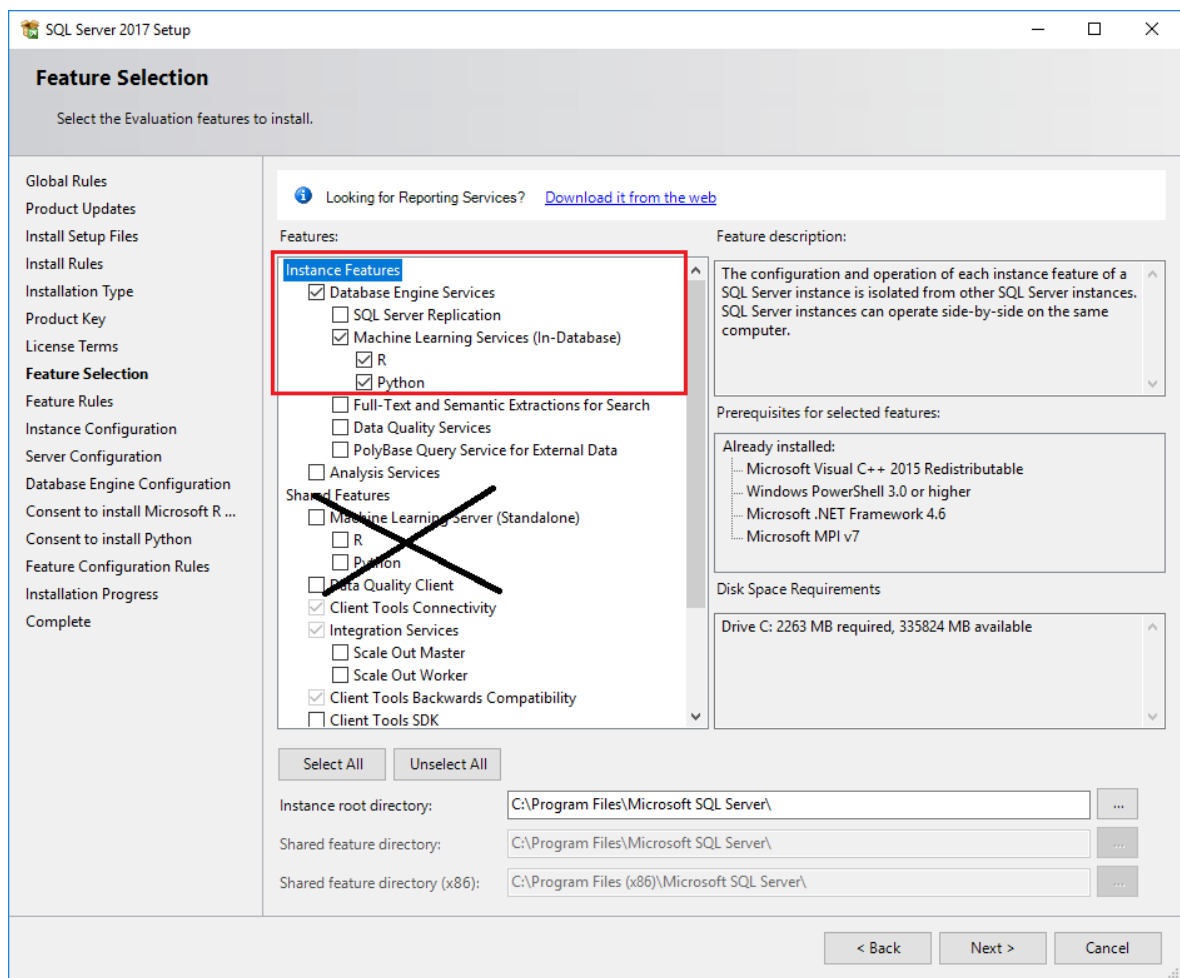
This option installs the database services that support R and Python script execution.

- **R**

Select this option to add the Microsoft R packages, interpreter, and open-source R.

- **Python**

Select this option to add the Microsoft Python packages, the Python 3.5 executable, and select libraries from the Anaconda distribution.



⚠ Note

Don't select the **Machine Learning Server (Standalone)** option under **Shared Features**. That option is intended for use on a separate computer.

4. On the **Consent to Install Microsoft R Open** page, select **Accept > Next**.

The license agreement covers:

- Microsoft R Open.
- Open-source R base packages and tools.
- Enhanced R packages and connectivity providers from the Microsoft development team.

5. On the **Consent to Install Python** page, select **Accept > Next**. The Python open-source license agreement also covers Anaconda and related tools, plus some new Python libraries from the Microsoft development team.

⚠ Note

If the computer that you're using doesn't have internet access, you can pause setup at this point to download the installers separately. For more information, see [Install machine learning components without internet access](#).

6. On the **Ready to Install** page, verify that these selections are included, and then select **Install**:

- Database Engine Services
- Machine Learning Services (in-database)
- R, Python, or both

Note the location of the folder under the path `..\Setup Bootstrap\Log` where the configuration files are stored. When setup is complete, you can review the installed components in the summary file.

7. After setup is complete, if you're instructed to restart the computer, do so. It's important to read the message from the Installation Wizard when you finish setup. For more information, see [View and read SQL Server Setup log files](#).

Set environment variables

For R feature integration only, you should set the `MKL_CBWR` environment variable to [ensure consistent output](#) from Intel Math Kernel Library (MKL) calculations:

1. In Control Panel, select **System and Security** > **System** > **Advanced System Settings** > **Environment Variables**.
2. Create a new user or system variable:
 - Set the variable name to `MKL_CBWR`.
 - Set the variable value to `AUTO`.

This step requires a server restart. If you're about to enable script execution, you can hold off on the restart until all of the configuration work is done.

Enable script execution

1. Use [SQL Server Management Studio \(SSMS\)](#) or [Azure Data Studio](#) to connect to the instance where you installed SQL Server Machine Learning Services.
 2. Select **New Query** to open a query window, and then run the following command:
-

```
SQL
```

```
EXEC sp_configure
```

- The value for the property `external scripts enabled` should be `0` at this point. The feature is turned off by default. To turn it on so you can run R or Python scripts, run the following statement:

```
SQL
```

```
EXEC sp_configure 'external scripts enabled', 1  
RECONFIGURE WITH OVERRIDE
```

If you've already enabled the feature for the R language, you don't need to run `RECONFIGURE` a second time for Python. The underlying extensibility platform supports both languages.

Restart the service

When the installation is complete, restart the database engine. Restarting the service also automatically restarts the related SQL Server Launchpad service.

You can restart the service by using any of these methods:

- The right-click **Restart** command for the instance in Object Explorer in SSMS
- The **Services** Microsoft Management Console (MMC) item in Control Panel
- [SQL Server Configuration Manager](#)

Verify installation

Use the following steps to verify that all components used to launch external scripts are running:

1. In SQL Server Management Studio, open a new query window and run the following command:

```
SQL
```

```
EXECUTE sp_configure 'external scripts enabled'
```

Then, `run_value` is set to `1`.

2. Open the **Services** control panel item or SQL Server Configuration Manager, and verify that **SQL Server Launchpad service** is running. You should have one service for every database engine instance that has R or Python installed. For more information about the service, see [Extensibility architecture in SQL Server Machine Learning Services](#).
3. If Launchpad is running, you can run simple Python and R scripts to verify that external scripting runtimes can communicate with SQL Server.

Open a new **Query** window in SQL Server Management Studio, and then run a script such as:

- For R:

```
SQL

EXEC sp_execute_external_script @language =N'R',
@script=N'
OutputDataSet <- InputDataSet;
',
@input_data_1 =N'SELECT 1 AS hello'
WITH RESULT SETS ([[hello] int not null]);
GO
```

- For Python:

```
SQL

EXEC sp_execute_external_script @language =N'Python',
@script=N'
OutputDataSet = InputDataSet;
',
@input_data_1 =N'SELECT 1 AS hello'
WITH RESULT SETS ([[hello] int not null]);
GO
```

The first time that the external script runtime is loaded, the script can take a little while to run. The results should be something like this:

hello
1

ⓘ **Note**

Columns or headings used in the Python script aren't returned automatically. To add column names for your output, you must specify the schema for the return data set. Do this by using the `WITH RESULTS` parameter of the stored procedure, naming the columns, and specifying the SQL data type.

For example, you can add the following line to generate an arbitrary column name: `WITH RESULT SETS ((Col1 AS int))`.

Apply updates

Existing installation

If you've added Machine Learning Services to an existing SQL Server instance and have previously applied a cumulative update (CU), the versions of your database engine and the Machine Learning Services feature might be different. This difference might result in unexpected behavior or errors because `launchpad.exe` and `sqlservr.exe` have different versions.

Follow these steps to bring Machine Learning Services to the same version as your database engine:

1. Determine the cumulative update that you have for the database engine. Run this T-SQL statement:

```
SQL
```

```
SELECT @@VERSION
```

Here's an example output from SQL Server 2019 CU 8:

```
Microsoft SQL Server 2019 (RTM-CU8-GDR) (KB4583459) - 15.0.4083.2 (X64)
Nov 2 2020 18:35:09 Copyright (C) 2019 Microsoft Corporation
Developer Edition (64-bit) on Windows 10 Enterprise 10.0 (X64) (Build
19042: ) (Hypervisor)
```

For more information, see [Determine the version, edition, and update level of SQL Server and its components](#).

2. If necessary, download the [cumulative update](#) that you installed for the database engine.

3. Run the installation of the cumulative update, and follow the instructions to install it for Machine Learning Services again. Select the existing instance where Machine Learning Services is installed. The upgrade status shows **Incompletely Installed** on the **Feature Selection** page.
4. Select **Next** and continue with installation.

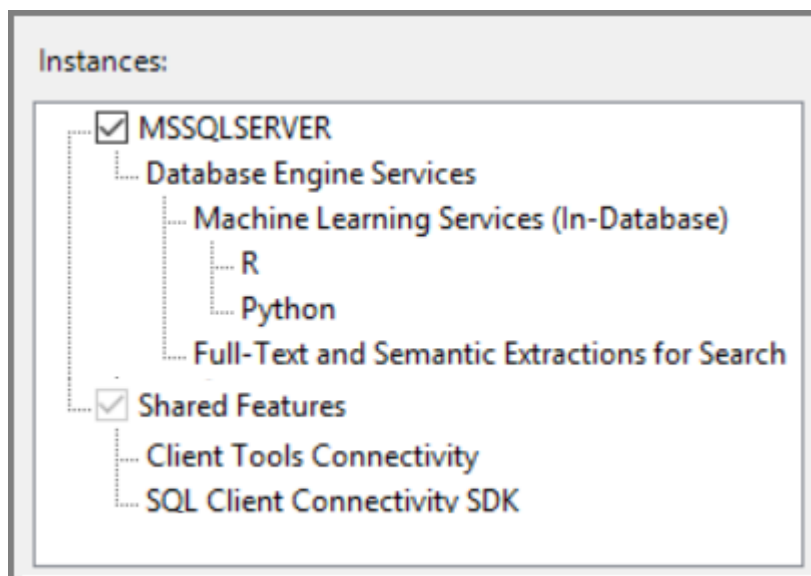
New installation

If you install Machine Learning Services with a new installation of the SQL Server database engine, we recommend that you apply the latest cumulative update to both the database engine and machine learning components.

On internet-connected devices, cumulative updates are typically applied through Windows Update. But you can also use the following steps for controlled updates. When you apply the update for the database engine, setup pulls cumulative updates for any Python or R features that you installed on the same instance.

Disconnected servers require extra steps. For more information, see [Install on computers with no internet access > Apply cumulative updates](#).

1. Start with a baseline instance already installed: SQL Server initial release.
2. Go to the [list of updates for Microsoft SQL Server](#).
3. Select the latest cumulative update. An executable is downloaded and extracted automatically.
4. Run Setup and accept the license terms.
5. On the **Feature selection** page, review the features for which cumulative updates are applied. You should see every feature installed for the current instance, including machine learning features. Setup downloads the CAB files that are necessary to update all features.



6. Continue through the wizard. Accept the license terms for R and Python distributions.

Additional configuration

If the external script verification step was successful, you can run R or Python commands from SQL Server Management Studio, Visual Studio Code, or any other client that can send T-SQL statements to the server.

If you got an error when you ran the command, you might need to make additional configurations to the service or database. At the instance level, additional configurations might include:

- [Configure a firewall for SQL Server Machine Learning Services](#)
- [Enable additional network protocols](#)
- [Enable remote connections](#)
- [Create a login for SQLRUserGroup](#)
- [Manage disk quotas](#) to prevent external scripts from running tasks that exhaust disk space

On the database, you might need configuration updates. For more information, see [Give users permission to SQL Server Machine Learning Services](#).

ⓘ Note

Whether the additional configuration is required depends on your security schema, where you installed SQL Server, and how you expect users to connect to the database and run external scripts.

Suggested optimizations

Now that you have everything working, you might also want to optimize the server to support machine learning or install a pre-trained machine learning model.

Add more worker accounts

If you expect many users to run scripts concurrently, you can increase the number of worker accounts that are assigned to the Launchpad service. For more information, see [Scale concurrent execution of external scripts in SQL Server Machine Learning Services](#).

Optimize the server for script execution

The default settings for SQL Server setup are intended to optimize the balance of the server for a variety of other services and applications.

Under the default settings, resources for machine learning are sometimes restricted or throttled, particularly in memory-intensive operations.

To ensure that machine learning jobs are prioritized and resourced appropriately, we recommend that you use SQL Server Resource Governor to configure an external resource pool. You might also want to change the amount of memory that's allocated to the SQL Server database engine, or increase the number of accounts that run under the SQL Server Launchpad service.

- To configure a resource pool for managing external resources, see [Create an external resource pool](#).
- To change the amount of memory reserved for the database, see [Server memory configuration options](#).
- To change the number of R accounts that SQL Server Launchpad can start, see [Scale concurrent execution of external scripts in SQL Server Machine Learning Services](#).

If you're using Standard Edition and don't have Resource Governor, you can use dynamic management views, SQL Server Extended Events, and Windows event monitoring to help manage the server resources.

Install additional Python and R packages

The Python and R solutions that you create for SQL Server can call:

- Basic functions.
- Functions from the proprietary packages installed with SQL Server.
- Third-party packages that are compatible with the version of open-source Python and R that SQL Server installs.

Packages that you want to use from SQL Server must be installed in the default library that the instance uses. If you have a separate installation of Python or R on the computer, or if you installed packages to user libraries, you can't use those packages from T-SQL.

To install and manage additional packages, you can set up user groups to share packages on a per-database level, or you can configure database roles to enable users to install their own packages. For more information, see [Install Python packages](#) and [Install new R packages](#).

Next steps

Python developers can learn how to use Python with SQL Server by following these tutorials:

- [Python tutorial: Predict ski rental with linear regression in SQL Server Machine Learning Services](#)
- [Python tutorial: Build a model to categorize customers with SQL machine learning](#)

R developers can get started with some simple examples and learn the basics of how R works with SQL Server. For your next step, see the following links:

- [Quickstart: Run R in T-SQL](#)
- [Tutorial: In-database analytics for R developers](#)

Offline install SQL Server Machine Learning Services on Windows computers with no internet access

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This article describes how to install SQL Server Machine Learning Services on Windows offline on computers with no internet access isolated behind a network firewall.

By default, installers connect to Microsoft download sites to get required and updated components for machine learning on SQL Server. If firewall constraints prevent the installer from reaching these sites, you can use an internet-connected device to download files, transfer files to an offline server, and then run setup.

Note

Feature capabilities and installation options vary between versions of SQL Server. Use the version selector dropdown to choose the appropriate version of SQL Server.

In-database analytics consist of database engine instance, plus additional components for R and Python integration, depending on the version of SQL Server.

- Beginning with SQL Server 2022 (16.x), runtimes for R, Python, and Java, are no longer installed with SQL Setup. Instead, install your desired R and/or Python custom runtime(s) and packages. The offline installation process is therefore similar to the online process. For more information, see [Install SQL Server 2022 Machine Learning Services on Windows](#) or [Install SQL Server 2022 Machine Learning Services on Linux](#).
- SQL Server 2019 includes R, Python, and Java.
- SQL Server 2017 includes R and Python.
- SQL Server 2016 is R-only.

On an isolated server, machine learning and R/Python language-specific features are added through CAB files.

SQL Server 2017 offline install

To install SQL Server Machine Learning Services (R and Python) on an isolated server, start by downloading the initial release of SQL Server and the corresponding CAB files for R and Python support. Even if you plan to immediately update your server to use the latest cumulative update, an initial release must be installed first.

ⓘ Note

SQL Server 2017 does not have service packs. It's the first release of SQL Server to use the initial release as the only base line, with servicing through cumulative updates only.

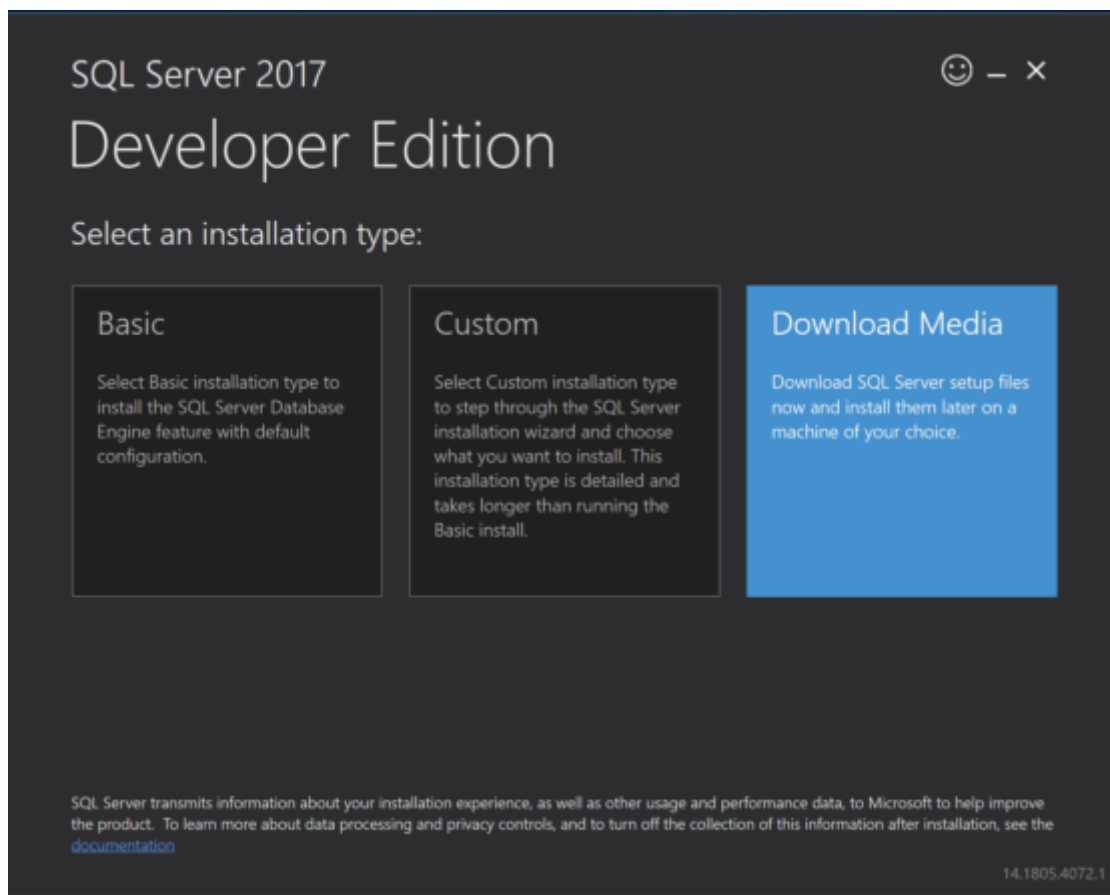
1 - Download 2017 CABs

On a computer having an internet connection, download the CAB files providing R and Python features for the initial release and the installation media for SQL Server 2017.

Release	Download link
Microsoft R Open	SRO_3.3.3.24_1033.cab ↗
Microsoft R Server	SRS_9.2.0.24_1033.cab ↗
Microsoft Python Open	SPO_9.2.0.24_1033.cab ↗
Microsoft Python Server	SPS_9.2.0.24_1033.cab ↗

2 - Get SQL Server 2017 installation media

1. On a computer having an internet connection, launch SQL Server 2017 Setup from your installation media.
2. Double-click setup and choose the **Download Media** installation type. With this option, setup creates a local .iso (or .cab) file containing the installation media.



Transfer files

Copy the SQL Server installation media (.iso or .cab) and in-database analytics CAB files to the target computer. Place the CAB files and installation media file in the same folder on the target machine, such as the setup user's %TEMP% folder.

The %TEMP% folder is required for Python CAB files. For R, you can use %TEMP% or set the `myrcachedirectory` parameter to the CAB path.

Run Setup

When you run SQL Server Setup on a computer disconnected from the internet, Setup adds an **Offline installation** page to the wizard so that you can specify the location of the CAB files you copied in the previous step.

1. To begin installation, double-click the .iso or .cab file to access the installation media. You should see the **setup.exe** file.
2. Right-click **setup.exe** and run as administrator.
3. When the setup wizard displays the licensing page for open-source R or Python components, click **Accept**. Acceptance of licensing terms allows you to proceed to the next step.

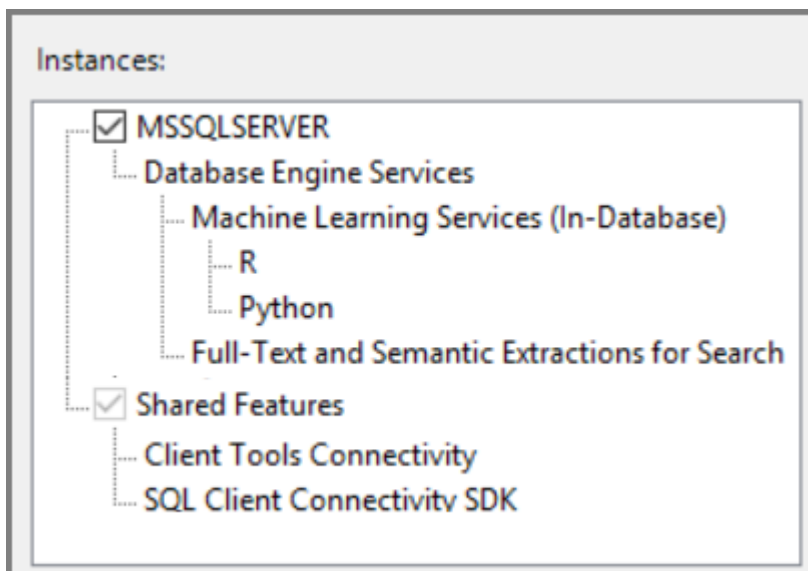
4. When you get to the **Offline installation** page, in **Install Path**, specify the folder containing the CAB files you copied earlier.
5. Continue following the on-screen prompts to complete the installation.

Apply cumulative updates

We recommend that you apply the latest cumulative update to both the database engine and machine learning components.

Cumulative updates are installed through the Setup program.

1. Start with a baseline instance. You can only apply cumulative updates to existing installations of the initial release of SQL Server.
2. On an internet connected device, go to the cumulative update list for your version of SQL Server. [See Determine the version, edition, and update level of SQL Server and its components.](#)
3. Select the latest cumulative update to download the executable.
4. Get corresponding CAB files for R and Python. For download links, see [CAB downloads for cumulative updates on SQL Server in-database analytics instances.](#)
5. Transfer all files, executable and CAB files, to the same folder on the offline computer.
6. Run Setup. Accept the licensing terms, and on the Feature selection page, review the features for which cumulative updates are applied. You should see every feature installed for the current instance, including machine learning features.



7. Continue through the wizard, accepting the licensing terms for R and Python distributions. During installation, you are prompted to choose the folder location containing the updated CAB files.

Set environment variables

For R feature integration only, you should set the `MKL_CBWR` environment variable to [ensure consistent output](#) from Intel Math Kernel Library (MKL) calculations.

1. In Control Panel, click **System and Security > System > Advanced System Settings > Environment Variables**.
2. Create a new User or System variable.
 - Set variable name to `MKL_CBWR`
 - Set the variable value to `AUTO`

This step requires a server restart. If you are about to enable script execution, you can hold off on the restart until all of the configuration work is done.

Post-install configuration

After installation is finished, restart the service and then configure the server to enable script execution:

- [Enable external script execution](#)

An initial offline installation of SQL Server Machine Learning Services requires the same configuration as an online installation:

- [Verify installation](#)
- [Additional configuration as needed](#)




Next steps

To use Machine Learning Services to execute Python and R scripts in-database, see:

- SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x): [Install SQL Server Machine Learning Services](#)
- SQL Server 2022 (16.x): [Install SQL Server 2022 Machine Learning Services \(Python and R\) on Windows](#) or [Install SQL Server Machine Learning Services \(Python and R\) on Linux](#)

CAB downloads for offline installation of cumulative updates for SQL Server Machine Learning Services

Article • 06/28/2023

Applies to:  SQL Server 2016 (13.x),  SQL Server 2017 (14.x), and  SQL Server 2019 (15.x)

Download Python and R CAB files for SQL Server Machine Learning Services. These CAB files contain updates to the Machine Learning Services (Python and R) feature and are used when installing SQL Server on a server without internet access.

This article lists download links to CAB files for each cumulative update. For more information about offline installs, see [Install SQL Server machine learning components without internet access](#).







This article applies to SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x).

Prerequisites

Start with a baseline installation. On SQL Server Machine Learning Services, the initial release is the baseline installation. You can also apply cumulative updates.

SQL Server 2017 CABs

CAB files are listed in reverse chronological order. When you download the CAB files and transfer them to the target computer, place them in a convenient folder such as **Downloads** or the setup user's %temp% folder.






Release	Component	Download link	Issues addressed
SQL Server 2017 CU29  - CU31 			
	Microsoft R Open	SRO_3.5.2.777_1033.cab 	
	R Server	SRS_9.4.7.1162_1033.cab 	
	Microsoft Python Open	SPO_4.5.12.479_1033.cab 	
	Python Server	SPS_9.4.7.1226_1033.cab 	Fixes <code>sp_execute_external_script</code> execution failures for Python by removing breaking numpy package version mismatch.

Release	Component	Download link	Issues addressed
SQL Server 2017 CU27 ↗ -CU28 ↗			
	Microsoft R Open	SRO_3.5.2.777_1033.cab ↗	
	R Server	SRS_9.4.7.1162_1033.cab ↗	
	Microsoft Python Open	SPO_4.5.12.479_1033.cab ↗	
	Python Server	SPS_9.4.7.1162_1033.cab ↗	
SQL Server 2017 CU22 ↗ -CU26 ↗			
	Microsoft R Open	SRO_3.5.2.777_1033.cab ↗	
	R Server	SRS_9.4.7.958_1033.cab ↗	
	Microsoft Python Open	SPO_4.5.12.479_1033.cab ↗	
	Python Server	SPS_9.4.7.958_1033.cab ↗	
SQL Server 2017 CU19 ↗ -CU20 ↗			
	Microsoft R Open	SRO_3.3.3.1900_1033.cab ↗	Fixes the bug where <code>sp_execute_external_script</code> executing an R script shows warning message
	R Server	SRS_9.2.0.1900_1033.cab ↗	No change from previous versions.
	Microsoft Python Open	SPO_9.2.0.1400_1033.cab ↗	No change from previous versions.
	Python Server	SPS_9.2.0.1900_1033.cab ↗	Fixes the bug where <code>sp_execute_external_script</code> executing a python script sometimes loses data when returning varbinary or binary data type back to SQL Server in the form of an OutputDataSet.
SQL Server 2017 CU14 ↗ -CU15 ↗ - CU16 ↗ -CU17 ↗ -CU18 ↗			





Release	Component	Download link	Issues addressed
	Microsoft R Open	SRO_3.3.3.1400_1033.cab	Binaries within the package are now signed.
	R Server	SRS_9.2.0.1400_1033.cab	Binaries within the package are now signed.
	Microsoft Python Open	SPO_9.2.0.1400_1033.cab	Binaries within the package are now signed.
	Python Server	SPS_9.2.0.1400_1033.cab	Binaries within the package are now signed.
SQL Server 2017 CU13			
	Microsoft R Open	SRO_3.3.3.1300_1033.cab	No change from previous versions.
	R Server	SRS_9.2.0.1300_1033.cab	Contains a fix for upgrading an operationalized standalone R Server , as installed through SQL Server Setup. Use the CU13 CABs and follow these instructions to apply the update.
	Microsoft Python Open	SPO_9.2.0.24_1033.cab	No change from previous versions.
	Python Server	SPS_9.2.0.1300_1033.cab	Contains a fix for upgrading an operationalized standalone Python Server , as installed through SQL Server Setup. Use the CU13 CABs and follow these instructions to apply the update.
SQL Server 2017 CU10 - CU11 - CU12			
	Microsoft R Open	SRO_3.3.3.300_1033.cab	No change from previous versions.
	R Server	SRS_9.2.0.1000_1033.cab	Minor fixes.
	Microsoft Python Open	SPO_9.2.0.24_1033.cab	No change from previous versions.

Release	Component	Download link	Issues addressed
	Python Server	SPS_9.2.0.1000_1033.cab	Python rx_data_step loses row order when duplicates are removed. SPEE fails datatype detection on clustered columnstore index. Returns an empty table when columns contain all null values.
SQL Server 2017 CU8 -CU9			
	Microsoft R Open	SRO_3.3.3.300_1033.cab	No change from previous versions.
	R Server	SRS_9.2.0.800_1033.cab	
	Microsoft Python Open	SPO_9.2.0.24_1033.cab	No change from previous versions.
	Python Server	SPS_9.2.0.800_1033.cab	
SQL Server 2017 CU6 -CU7			
	Microsoft R Open	SRO_3.3.3.300_1033.cab	No change from previous versions.
	R Server	SRS_9.2.0.600_1033.cab	
	Microsoft Python Open	SPO_9.2.0.24_1033.cab	No change from previous versions.
	Python Server	SPS_9.2.0.600_1033.cab	DateTime data types in SPEES query. improved error messages in microsoftml when pre-trained models are missing. Fixes to revoscalepy transform functions and variables.
SQL Server 2017 CU5			
	Microsoft R Open	SRO_3.3.3.300_1033.cab	No change from previous versions.
	R Server	SRS_9.2.0.500_1033.cab	Long path-related errors in rxInstallPackages. Connections in a loopback for RxExec.

Release	Component	Download link	Issues addressed
	Microsoft Python Open	No change from previous versions.	
	Python Server	SPS_9.2.0.500_1033.cab	Connections in a loopback for rx_exec.
SQL Server 2017 CU4			
	Microsoft R Open	SRO_3.3.3.300_1033.cab	No change from previous versions.
	R Server	SRS_9.2.0.400_1033.cab	
	Microsoft Python Open	SPO_9.2.0.24_1033.cab	No change from previous versions.
	Python Server	SPS_9.2.0.400_1033.cab	
SQL Server 2017 CU3			
	Microsoft R Open	SRO_3.3.3.300_1033.cab	
	R Server	SRS_9.2.0.300_1033.cab	
	Microsoft Python Open	SPO_9.2.0.24_1033.cab	No change from previous versions.
	Python Server	SPS_9.2.0.300_1033.cab	Python model serialization in revoscalepy, using the rx_serialize_model function . Native scoring support, plus enhancements to real-time scoring .
SQL Server 2017 CU1 -CU2			
	Microsoft R Open	SRO_3.3.3.24_1033.cab	No change from previous versions.
	R Server	SRS_9.2.0.100_1033.cab	
	Microsoft Python Open	SPO_9.2.0.24_1033.cab	No change from previous versions.

Release	Component	Download link	Issues addressed
	Python Server	SPS_9.2.0.100_1033.cab 	Adds rx_create_col_info for returning schema information. Enhancements to rx_exec to support parallel scenarios using the <code>RxLocalParallel</code> compute context.
Initial release			
	Microsoft R Open	SRO_3.3.3.24_1033.cab 	
	R Server	SRS_9.2.0.24_1033.cab 	
	Microsoft Python Open	SPO_9.2.0.24_1033.cab 	
	Python Server	SPS_9.2.0.24_1033.cab 	

GDR releases of SQL Server will require the same component .cab file versions as the next earlier non-GDR release, such as a CU.

Release	Component	Download link
SQL 2017 GDR		
	Microsoft R Open	SRO_3.3.3.1400_1033.cab 
	R Server	SRS_9.2.0.1400_1033.cab 
	Microsoft Python Open	SPO_9.2.0.1400_1033.cab 
	Python Server	SPS_9.2.0.1400_1033.cab 

Next steps

- [Apply cumulative updates on computers without internet access](#)
- [Apply cumulative updates on computers having internet connectivity](#)
- [Apply cumulative updates to a standalone server](#)

Install SQL Server Machine Learning Services with R and Python from the command line

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This article provides instructions for installing [SQL Server Machine Learning Services](#) with Python and R from a command line.

You can specify silent, basic, or full interaction with the Setup user interface. This article supplements [Install SQL Server from the Command Prompt](#), covering the parameters unique to R and Python machine learning components.

Note

Feature capabilities and installation options vary between versions of SQL Server. Use the version selector dropdown to choose the appropriate version of SQL Server.

Pre-install checklist

- Run commands from an elevated command prompt.
- A database engine instance is required for in-database installations. You cannot install just R or Python features, although you can [add them incrementally to an existing instance](#). If you want just R and Python without the database engine, install the [standalone server](#).
- Do not install on a failover cluster. The security mechanism used for isolating R and Python processes is not compatible with a Windows Server failover cluster environment.
- Do not install on a domain controller. The Machine Learning Services portion of setup will fail.
- Avoid installing standalone and in-database instances on the same computer. A standalone server will compete for the same resources, undermining the performance of both installations.

Command line arguments

The `/FEATURES` argument is required, as are licensing term agreements.

When installing through the command prompt, SQL Server supports full quiet mode by using the `/Q` parameter, or Quiet Simple mode by using the `/QS` parameter. The `/QS` switch only shows progress, does not accept any input, and displays no error messages if encountered. The `/QS` parameter is only supported when `/Action=install` is specified.

Command line arguments for SQL Server 2017

Arguments	Description
<code>/FEATURES = AdvancedAnalytics</code>	Installs the in-database version: SQL Server Machine Learning Services (In-Database).
<code>/FEATURES = SQL_INST_MR</code>	Pair this with AdvancedAnalytics. Installs the (In-Database) R feature, including Microsoft R Open and the proprietary R packages.
<code>/FEATURES = SQL_INST_MPY</code>	Pair this with AdvancedAnalytics. Installs the (In-Database) Python feature, including Anaconda and the proprietary Python packages.
<code>/FEATURES = SQL_SHARED_MR</code>	Installs the R feature for the standalone version: SQL Server Machine Learning Server (Standalone). A standalone server is a "shared feature" not bound to a database engine instance.
<code>/FEATURES = SQL_SHARED_MPY</code>	Installs the Python feature for the standalone version: SQL Server Machine Learning Server (Standalone). A standalone server is a "shared feature" not bound to a database engine instance.
<code>/IACCEPTROPENLICENSETERMS</code>	Indicates you have accepted the license terms for using the open source R components.
<code>/IACCEPTPYTHONLICENSETERMS</code>	Indicates you have accepted the license terms for using the Python components.
<code>/IACCEPTSQLSERVERLICENSETERMS</code>	Indicates you have accepted the license terms for using SQL Server.
<code>/MRCACHEDIRECTORY</code>	For offline setup, sets the folder containing the R component CAB files.

Arguments	Description
/MPYCACHEDIRECTORY	Reserved for future use. Use %TEMP% to store Python component CAB files for installation on computers that do not have an internet connection.

In-database instance installations

In-database analytics are available for database engine instances, required for adding the **AdvancedAnalytics** feature to your installation. You can install a database engine instance with advanced analytics, or [add it to an existing instance](#).

To view progress information without the interactive on-screen prompts, use the /qs argument.

Important

After installation, two additional configuration steps remain. Integration is not complete until these tasks are performed. See [Post-installation configuration](#) for instructions.

SQL Server Machine Learning Services: database engine, advanced analytics with Python and R

For a concurrent installation of the database engine instance, provide the instance name and an administrator (Windows) login. Include features for installing core and language components, as well as acceptance of all licensing terms.

Windows Command Prompt

```
Setup.exe /qs /ACTION=Install  
/FEATURES=SQLEngine,ADVANCEDANALYTICS,SQL_INST_MR,SQL_INST_MPY  
/INSTANCENAME=MSSQLSERVER /SQLSYSADMINACCOUNTS="<Windows-username>"  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPTOPENLICENSETERMS  
/IACCEPTPYTHONLICENSETERMS
```

This the same command, but with a SQL Server login on a database engine using mixed authentication.

Windows Command Prompt


```
Setup.exe /q /ACTION=Install  
/FEATURES=SQLEngine,ADVANCEDANALYTICS,SQL_INST_MR,SQL_INST_MPY  
/INSTANCENAME=MSSQLSERVER /SECURITYMODE=SQL /SAPWD="%password%"  
/SQLSYSADMINACCOUNTS="<sql-username>"  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPTROPENLICENSETERMS  
/IACCEPTPYTHONLICENSETERMS
```

This example is Python only, showing that you can add one language by omitting a feature.

Windows Command Prompt

```
Setup.exe /qs /ACTION=Install  
/FEATURES=SQLEngine,ADVANCEDANALYTICS,SQL_INST_MPY  
/INSTANCENAME=MSSQLSERVER /SQLSYSADMINACCOUNTS="<username>"  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPTPYTHONLICENSETERMS
```

Post-installation configuration (required)

Applies to in-database installations only.

When SQL Setup for SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x) is finished, you have a database engine instance with R and Python, the Microsoft R and Python packages, Microsoft R Open, Anaconda, tools, samples, and scripts that are part of the distribution.

Beginning with SQL Server 2022 (16.x), runtimes for R, Python, and Java, are no longer installed with SQL Setup. Instead, install your desired R and/or Python custom runtime(s) and packages. For more information, see [Install SQL Server 2022 Machine Learning Services on Windows](#) or [Install SQL Server Machine Learning Services \(Python and R\) on Linux](#).

Two more tasks are required to complete the installation:

1. Restart the database engine service.
2. SQL Server Machine Learning Services: Enable external scripts before you can use the feature. Follow the instructions in [Install SQL Server Machine Learning Services \(In-Database\)](#) as your next step.

Add advanced analytics to an existing database engine instance

When adding in-database advanced analytics to an existing database engine instance, provide the instance name. For example, if you previously installed a SQL Server 2017 or later database engine and Python, you could use this command to add R.

Windows Command Prompt

```
Setup.exe /qs /ACTION=Install /FEATURES=SQL_INST_MR  
/INSTANCENAME=MSSQLSERVER  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPTROPENLICENSETERMS
```

Silent install

A silent installation suppresses the check for .cab file locations. For this reason, you must specify the location where .cab files are to be unpacked. For Python, CAB files must be located in %TEMP*. For R, you can set the folder path using you can the temp directory for this.

Windows Command Prompt

```
Setup.exe /q /ACTION=Install  
/FEATURES=SQLEngine,ADVANCEDANALYTICS,SQL_INST_MR,SQL_INST_MPY  
/INSTANCENAME=MSSQLSERVER /SQLSYSADMINACCOUNTS="<username>"  
/IACCEPTSQLSERVERLICENSETERMS /IACCEPTROPENLICENSETERMS  
/IACCEPTPYTHONLICENSETERMS  
/MRCACHEDIRECTORY=%temp%
```

Standalone server installations

Important

The support for Machine Learning Server (previously known as R Server) ended on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Applies to: SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x) only.

A standalone server is a "shared feature" not bound to a database engine instance. The following examples show valid syntax for installation of the standalone server.

SQL Server Machine Learning Server supports Python and R on a standalone server:

Windows Command Prompt

```
Setup.exe /q /ACTION=Install /FEATURES=SQL_SHARED_MR,SQL_SHARED_MPY  
/IACCEPTROPENLICENSETERMS /IACCEPTPYTHONLICENSETERMS  
/IACCEPTSQLSERVERLICENSETERMS
```

When SQL Setup for SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x) is finished, you have a server, Microsoft packages, open-source distributions of R and Python, tools, samples, and scripts that are part of the distribution.

Beginning with SQL Server 2022 (16.x), runtimes for R, Python, and Java, are no longer installed with SQL Setup. Instead, install your desired R and/or Python custom runtime(s) and packages. For more information, see [Install SQL Server 2022 Machine Learning Services on Windows](#) or [Install SQL Server Machine Learning Services \(Python and R\) on Linux](#).

To open an R console window, go to `\Program files\Microsoft SQL Server\150(or 140,130)\R_SERVER\bin\x64` and double-click **RGui.exe**. New to R? Try this tutorial: [Basic R commands and RevoScaleR functions: 25 common examples](#).

To open a Python command, go to `\Program files\Microsoft SQL Server\150 (or 140)\PYTHON_SERVER\bin\x64` and double-click **python.exe**.

Next steps

Python developers can learn how to use Python with SQL Server by following these tutorials:




- [Python tutorial: Predict ski rental with linear regression in SQL Server Machine Learning Services](#)
- [Python tutorial: Categorizing customers using k-means clustering with SQL Server Machine Learning Services](#)

R developers can get started with some simple examples, and learn the basics of how R works with SQL Server. For your next step, see the following links:

- [Quickstart: Run R in T-SQL](#)
- [Tutorial: In-database analytics for R developers](#)

Install pre-trained machine learning models on SQL Server

Article • 03/17/2023

Applies to:  SQL Server 2016 (13.x),  SQL Server 2017 (14.x), and  SQL Server 2019 (15.x)

This article applies to SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x).

This article explains how to use PowerShell to add free pre-trained machine learning models for *sentiment analysis* and *image featurization* to a SQL Server instance having R or Python integration. The pre-trained models are built by Microsoft and ready-to-use, added to an instance as a post-install task. For more information about these models, see the [Resources](#) section of this article.

Beginning with SQL Server 2022 (16.x), runtimes for R, Python, and Java, are no longer installed with SQL Setup. Instead, install your desired R and/or Python custom runtime(s) and packages. For more information, see [Install SQL Server 2022 Machine Learning Services \(Python and R\) on Windows](#).

Once installed, the pre-trained models are considered an implementation detail that power specific functions in the MicrosoftML (R) and microsoftml (Python) libraries. You should not (and cannot) view, customize, or retrain the models, nor can you treat them as an independent resource in custom code or paired other functions.

To use the pretrained models, call the functions listed in the following table.

R function (MicrosoftML)	Python function (microsoftml)	Usage
getSentiment	get_sentiment	Generates positive-negative sentiment score over text inputs.
featurizeImage	featurize_image	Extracts text information from image file inputs.

Prerequisites

Machine learning algorithms are computationally intensive. We recommend 16 GB RAM for low-to-moderate workloads, including completion of the tutorial walkthroughs using all of the sample data.

You must have administrator rights on the computer and SQL Server to add pre-trained models.

External scripts must be enabled and SQL Server LaunchPad service must be running. Installation instructions provide the steps for enabling and verifying these capabilities.

Download and install the latest cumulative update for your version of SQL Server. See the [Latest updates for Microsoft SQL Server](#).

[MicrosoftML R package](#) or [microsoftml Python package](#) contain the pre-trained models.

[SQL Server Machine Learning Services](#) includes both language versions of the machine learning library, so this prerequisite is met with no further action on your part. Because the libraries are present, you can use the PowerShell script described in this article to add the pre-trained models to these libraries.

Check whether pre-trained models are installed

The install paths for R and Python models are as follows:

- For R: `C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library\MicrosoftML\mxLibs\x64`
- For Python: `C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\Lib\site-packages\microsoftml\mxLibs`

Model file names are listed below:

- AlexNet_Updated.model
- ImageNet1K_mean.xml
- pretrained.model
- ResNet_101_Updated.model
- ResNet_18_Updated.model
- ResNet_50_Updated.model

If the models are already installed, skip ahead to the [validation step](#) to confirm availability.

Download the installation script

Visit <https://aka.ms/mlm4sql> to download the file `Install-MLModels.ps1`.

Execute with elevated privileges

1. Start PowerShell. On the task bar, right-click the PowerShell program icon and select **Run as administrator**.
2. The recommended execution policy during installation is "RemoteSigned". For more information on setting the PowerShell execution policy, see [Set-ExecutionPolicy](#). For example:

```
PowerShell

Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

3. Enter a fully-qualified path to the installation script file and include the instance name. Assuming the Downloads folder and a default instance, the command might look like this:

```
PowerShell

PS C:\WINDOWS\system32> C:\Users\\Downloads\Install-MLModels.ps1 MSSQLSERVER
```

Output

On an internet-connected SQL Server Machine Learning Services default instance with R and Python, you should see messages similar to the following.

```
PowerShell

MSSQL14.MSSQLSERVER
  Verifying R models [9.2.0.24]
  Downloading R models [C:\Users\\AppData\Local\Temp]
  Installing R models [C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER\R_SERVICES\]
  Verifying Python models [9.2.0.24]
  Installing Python models [C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\]
PS C:\WINDOWS\system32>
```

Verify installation

First, check for the new files in the [mxlibs folder](#). Next, run demo code to confirm the models are installed and functional.

R verification steps

1. Start **RGUI.EXE** at C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\bin\x64.
2. Paste in the following R script at the command prompt.

```
R

# Create the data
CustomerReviews <- data.frame(Review = c(
  "I really did not like the taste of it",
  "It was surprisingly quite good!",
  "I will never ever ever go to that place again!!"),
  stringsAsFactors = FALSE)

# Get the sentiment scores
sentimentScores <- rxFeaturize(data = CustomerReviews,
                               mlTransforms = getSentiment(vars =
list(SentimentScore = "Review")))

# Let's translate the score to something more meaningful
sentimentScores$PredictedRating <-
  ifelse(sentimentScores$SentimentScore > 0.6,
         "AWESOMENESS", "BLAH")

# Let's look at the results
sentimentScores
```

3. Press Enter to view the sentiment scores. Output should be as follows:

```
R

> sentimentScores

              Review SentimentScore
1 I really did not like the taste of it 0.4617899
2 It was surprisingly quite good!      0.9601924
3 I will never ever ever go to that place again!! 0.3103435
PredictedRating
1 BLAH
2 AWESOMENESS
3 BLAH
```

Python verification steps

1. Start **Python.exe** at C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES.

2. Paste in the following Python script at the command prompt

Python

```
import numpy
import pandas
from microsoftml import rx_logistic_regression, rx_featurize,
rx_predict, get_sentiment

# Create the data
customer_reviews = pandas.DataFrame(data=dict(review=[
    "I really did not like the taste of it",
    "It was surprisingly quite good!",
    "I will never ever ever go to that place again!!"]))

# Get the sentiment scores
sentiment_scores = rx_featurize(
    data=customer_reviews,
    ml_transforms=[get_sentiment(cols=dict(scores="review"))])

# Let's translate the score to something more meaningful
sentiment_scores["eval"] = sentiment_scores.scores.apply(
    lambda score: "AWESOMENESS" if score > 0.6 else "BLAH")
print(sentiment_scores)
```

3. Press Enter to print the scores. Output should be as follows:

Python

```
>>> print(sentiment_scores)
                                     review  scores
eval
0          I really did not like the taste of it  0.461790
BLAH
1          It was surprisingly quite good!  0.960192
AWESOMENESS
2  I will never ever ever go to that place again!!  0.310344
BLAH
>>>
```

ⓘ Note

If demo scripts fail, check the file location first. On systems having multiple instances of SQL Server, or for instances that run side-by-side with standalone versions, it's possible for the installation script to mis-read the environment and place the files in the wrong location. Usually, manually copying the files to the correct mxlib folder fixes the problem.

Examples using pre-trained models

The following link include example code invoking the pretrained models.

- [Code sample: Sentiment Analysis using Text Featurizer](#) ↗

Research and resources

Currently the models that are available are deep neural network (DNN) models for sentiment analysis and image classification. All pre-trained models were trained by using Microsoft's [Computation Network Toolkit](#) ↗, or CNTK.

The configuration of each network was based on the following reference implementations:

- ResNet-18
- ResNet-50
- ResNet-101
- AlexNet

For more information about the algorithms used in these deep learning models, and how they are implemented and trained using CNTK, see these articles:



- [Microsoft Researchers' Algorithm Sets ImageNet Challenge Milestone](#) ↗
- [Microsoft Computational Network Toolkit offers most efficient distributed deep learning computational performance](#) ↗

See also

- [SQL Server Machine Learning Services](#)
- [Upgrade R and Python components in SQL Server instances](#)
- [MicrosoftML package for R](#)
- [microsoftml package for Python](#)

Change the default R or Python language runtime version

Article • 03/17/2023

Applies to:  SQL Server 2016 (13.x)  SQL Server 2017 (14.x)

This article describes how to change the default version of R or Python used in [SQL Server 2016 R Services](#) or [SQL Server 2017 Machine Learning Services](#).

The following lists the versions of the R and Python runtime that are included in the different SQL Server versions.

SQL Server version	Service	Cumulative Update	R runtime versions	Python runtime version
SQL Server 2016	R Services	RTM - SP2 CU13	3.2.2	Not available
SQL Server 2016	R Services	SP2 CU14 and later	3.2.2 and 3.5.2	Not available
SQL Server 2017	Machine Learning Services	RTM - CU21	3.3.3	3.5.2
SQL Server 2017	Machine Learning Services	CU22 and later	3.3.3 and 3.5.2	3.5.2 and 3.7.2

Prerequisites

You need to install a Cumulative Update (CU) to change the default R or Python language runtime version:

- **SQL Server 2016:** Services Pack (SP) 2 Cumulative Update (CU) 14 or later
- **SQL Server 2017:** Cumulative Update (CU) 22 or later

To download the latest Cumulative Update, see the [Latest updates for Microsoft SQL Server](#).

Note

If you slipstream the Cumulative Update with a new installation of SQL Server, only the newest versions of the R and Python runtime will be installed.

Change R runtime version

If you have installed one of the above Cumulative Updates for SQL Server 2016 or 2017, you may have multiple versions of R in a SQL instance. Each version is contained in a subfolder of the instance folder with the name `R_SERVICES.<major>.<minor>` (the folder from the original installation may not have a version number appended to the folder name).

If you install a CU containing R 3.5, the new `R_SERVICES` folder is:

- SQL Server 2016: `C:\Program Files\Microsoft SQL Server\MSSQL13.<INSTANCE_NAME>\R_SERVICES.3.5`
- SQL Server 2017: `C:\Program Files\Microsoft SQL Server\MSSQL14.<INSTANCE_NAME>\R_SERVICES.3.5`

Each SQL instance uses one of these versions as the default version of R. You can change the default version by using the **RegisterRext.exe** command-line utility. The utility is located under the R folder in each SQL instance:

```
<SQL instance path>\R_SERVICES.n.n\library\RevoScaleR\rxLibs\x64\RegisterRext.exe
```

ⓘ Note

The functionality described in this article is available only with the copy of **RegisterRext.exe** included in SQL CUs. Don't use the copy that came with the original SQL installation.

To change the R runtime version, pass the following command line arguments to **RegisterRext.exe**:

- `/configure` - Required, specifies that you're configuring the default R version.
- `/instance: <instance name>` - Optional, the instance you want to configure. If not specified, the default instance is configured.
- `/rhome: <path to the R_SERVICES[n.n] folder>` - Optional, path to the runtime version folder you want to set as the default R version.

If you don't specify `/rhome`, the path configured is the path under which **RegisterRext.exe** is located.

Examples

Below are examples on how to change the R runtime version in SQL Server 2016 and 2017.

Change R runtime version in SQL Server 2016

For example, to configure R 3.5 as the default version of R for the instance MSSQLSERVER01 on SQL Server 2016:

Windows Command Prompt

```
cd "C:\Program Files\Microsoft SQL
Server\MSSQL13.MSSQLSERVER01\R_SERVICES.3.5\library\RevoScaleR\rxLibs\x64"

.\RegisterRext.exe /configure /rhome:"C:\Program Files\Microsoft SQL
Server\MSSQL13.MSSQLSERVER01\R_SERVICES.3.5" /instance:MSSQLSERVER01
```

Change R runtime version in SQL Server 2017

For example, to configure R 3.5 as the default version of R for the instance MSSQLSERVER01 on SQL Server 2017:

Windows Command Prompt

```
cd "C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER01\R_SERVICES.3.5\library\RevoScaleR\rxLibs\x64"

.\RegisterRext.exe /configure /rhome:"C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER01\R_SERVICES.3.5" /instance:MSSQLSERVER01
```

In these examples, you don't need to include the `/rhome` argument since you're specifying the same folder where **RegisterRext.exe** is located.

Change Python runtime version

If you have installed CU22 or later for SQL Server 2017, you may have multiple versions of Python in a SQL instance. Each version is contained in a subfolder of the instance folder with the name `PYTHON_SERVICES.<major>.<minor>` (the folder from the original installation may not have a version number appended to the folder name).

For example, if you install a CU containing Python 3.7, a new `PYTHON_SERVICES` folder is created:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.<INSTANCE_NAME>\PYTHON_SERVICES.3.7
```

Each SQL instance uses one of these versions as the default version of Python. You can change the default version by using the **RegisterRExt.exe** command-line utility. The utility is located under the Python folders in each SQL instance:

```
<SQL instance path> \PYTHON_SERVICES.n.n\Lib\site-packages\revoscalepy\rxLibs\RegisterRExt.exe
```

ⓘ Note

The functionality described in this article is available only with the copy of **RegisterRExt.exe** included in SQL CUs. Don't use the copy that came with the original SQL installation.

To change the Python runtime version, pass the following command line arguments to **RegisterRExt.exe**:

- `/configure` - Required, specifies that you're configuring the default Python version.
- `/python` - Specifies that you're configuring the default Python version. Optional if you specify `/pythonhome`.
- `/instance: <instance name>` - Optional, the instance you want to configure. If not specified, the default instance is configured.
- `/pythonhome: <path to the PYTHON_SERVICES[n.n] folder>` - Optional, path to the runtime version folder you want to set as the default Python version.

If you don't specify `/pythonhome`, the path configured is the path under which **RegisterRExt.exe** is located.

Example

For example, to configure **Python 3.7** as the default version of Python for the instance **MSSQLSERVER01** on SQL Server 2017:

Windows Command Prompt

```
cd "C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER01\PYTHON_SERVICES.3.7\Lib\site-
packages\revoscalepy\rxLibs"

.\RegisterRExt.exe /configure /pythonhome:"C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES.3.7" /instance:MSSQLSERVER01
```

In this example, you don't need to include the `/pythonhome` argument since you're specifying the same folder where **RegisterRext.exe** is located.

Remove a runtime version

To remove a version of R or Python, use **RegisterRExt.exe** with the `/cleanup` command-line argument, using the same `/rhome`, `/pythonhome`, and `/instance` arguments described previously.

For example, to remove the **R 3.2** folder from the instance MSSQLSERVER01:

Windows Command Prompt

```
.\RegisterRext.exe /cleanup /rhome:"C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER01\R_SERVICES" /instance:MSSQLSERVER01
```

For example, to remove the **Python 3.7** folder from the instance MSSQLSERVER01:

Windows Command Prompt

```
.\RegisterRExt.exe /cleanup /python /pythonhome:"C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER01\PYTHON_SERVICES.3.7" /instance:MSSQLSERVER01
```

RegisterRext.exe will ask you to confirm the clean up of the specified R runtime:

Are you sure you want to permanently delete the given runtime along with all the packages installed on it? [Yes(Y)/No(N)/Default(Yes)]:

To confirm, answer `Y` or press enter. Alternatively, you can skip this prompt by passing in `/y` or `/Yes` along the `/cleanup` option.

ⓘ Note

You can remove a version only if it's not configured as the default and it's not currently being used to run **RegisterRext.exe**.



Next steps

- [Get R package information](#)
- [Get Python package information](#)

- Install packages with R tools
- Install packages with Python tools

Upgrade Python and R runtime with binding in SQL Server Machine Learning Services

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x)  SQL Server 2017 (14.x)

Important

The support for Machine Learning Server (previously known as R Server) ended on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)


This article describes how to use an installation process called **binding** to upgrade the R or Python runtimes in [SQL Server 2016 R Services](#) or [SQL Server 2017 Machine Learning Services](#). You can get [newer versions of Python and R](#) by *binding* to [Microsoft Machine Learning Server](#).

Important

This article describes an old method for upgrading the R and Python runtimes, called *binding*. If you have installed **Cumulative Update (CU) 14 or later for SQL Server 2016 Services Pack (SP) 2** or **Cumulative Update (CU) 22 or later for SQL Server 2017**, see how to [change the default R or Python language runtime to a later version](#) instead.

What is binding?

Binding is an installation process that replaces the contents of your **R_SERVICES** and **PYTHON_SERVICES** folders with newer executables, libraries, and tools from [Microsoft Machine Learning Server](#).

The uploaded components included with the servicing model has changed. The service updates match the [support Timeline for Microsoft R Server & Machine Learning Server](#) on the [Modern Lifecycle](#) .

Except for component versions and service updates, binding doesn't change the basics of your installation:

- Python and R integration is still part of a database engine instance.
- Licensing is unchanged (no additional costs associated with binding).
- SQL Server support policies still hold for the database engine.

The rest of this article explains the binding mechanism and how it works for each version of SQL Server.

ⓘ Note

Binding applies to in-database instances only that are bound to SQL Server instances. In this case binding is not necessary for a Standalone installation.

Version map

The following tables are version maps. Each map shows package versions across releases. You can review upgrade paths when you bind to Microsoft Machine Learning Server (previously known as R Server, before the addition of Python support starting in Machine Learning Server 9.2.1).

The binding doesn't guarantee the latest version of R or Anaconda. When you bind to Microsoft Machine Learning Server, you get the R or Python version installed through Setup, which may not be the latest version available on the web.

SQL Server 2017 Machine Learning Services

Component	Initial Release	Machine Learning Server 9.3	Machine Learning Server 9.4.7
Microsoft R Open (MRO) over R	R 3.3.3	R 3.4.3	R 3.5.2
RevoScaleR	9.2	9.3	9.4.7
MicrosoftML	9.2	9.3	9.4.7
sqlrutils	1.0	1.0	1.0
olapR	1.0	1.0	1.0
Anaconda 4.2 over Python 3.5	4.2/3.5.2	4.2/3.5.2	
revoscalepy	9.2	9.3	9.4.7
microsoftml	9.2	9.3	9.4.7

Component	Initial Release	Machine Learning Server 9.3	Machine Learning Server 9.4.7
pretrained models	9.2	9.3	9.4.7

How component upgrade works

Executable files, Python, and R libraries are upgraded when you bind an existing installation of Python and R to Machine Learning Server.

Binding is executed by the [Microsoft Machine Learning Server installer](#) when you run Setup on an existing SQL Server database engine instance having Python or R integration.

Setup detects the existing features and prompts you to rebind to Machine Learning Server.

During binding, the contents of `C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES` and `\PYTHON_SERVICES` is overwritten with the newer executable files and libraries of `C:\Program Files\Microsoft\ML Server\R_SERVER` and `\PYTHON_SERVER`.

Binding applies to Python and R features only. Open-source packages for Python and R consists of:

- Anaconda
- Microsoft R Open
- Proprietary packages RevoScaleR
- Revoscalepy

The binding doesn't change the support model for the database engine instance or the version of SQL Server.

Binding is reversible. You can revert to SQL Server servicing by [unbinding the instance](#) and repairing your SQL Server database engine instance.

Bind to Machine Learning Server using Setup

Follow the steps to bind SQL Server to Microsoft Machine Learning Server using setup.

1. In SSMS, run `SELECT @@version` to verify the server meets minimum build requirements.

For SQL Server 2016 R Services, the minimum is [Service Pack 1](#) and [CU3](#).

2. Check the version of R base and RevoScaleR packages to confirm the existing versions are lower than what you plan to replace them with.

SQL

```
EXECUTE sp_execute_external_script
@language=N'R'
,@script = N'str(OutputDataSet);
packagematrix <- installed.packages();
Name <- packagematrix[,1];
Version <- packagematrix[,3];
OutputDataSet <- data.frame(Name, Version);'
, @input_data_1 = N''
WITH RESULT SETS ((PackageName nvarchar(250), PackageVersion
nvarchar(max) ))
```

3. Close SSMS and any other tools having an open connection to SQL Server. Binding overwrites program files. If SQL Server has open sessions, binding will fail with bind error code 6.
4. Download Microsoft Machine Learning Server onto the computer that has the instance you want to upgrade. We recommend the [latest version](#).
5. Unzip the folder and start ServerSetup.exe, located under MLSWIN93.
6. On **Configure the installation**, confirm the components to upgrade, and review the list of compatible instances.
7. On the **License agreement** page, select **I accept these terms** to accept the licensing terms for Machine Learning Server.
8. On successive pages, provide consent to additional licensing conditions for any open-source components you selected, such as Microsoft R Open or the Python Anaconda distribution.
9. On the **Almost there** page, make a note of the installation folder. The default folder is \Program Files\Microsoft\ML Server.

If you want to change the installation folder, select **Advanced** to return to the first page of the wizard. However, you must repeat all previous selections.

If upgrade fails, check [SqlBindR error codes](#) for more information.

Offline binding (no internet access)

For systems with no internet connectivity, you can download the installer and .cab files to an internet-connected machine, and then transfer files to the isolated server.

The installer (ServerSetup.exe) includes the Microsoft packages (RevoScaleR, MicrosoftML, olapR, sqlRUtils). The .cab files provide other core components. For example, the "SRO" cab provides R Open, Microsoft's distribution of open-source R.

The following instructions explain how to place the files for an offline installation.

1. Download the MLSWIN93 Installer. It downloads as a single zipped file. We recommend the [latest version](#), but you can also install [earlier versions](#).
2. Download .cab files. The following links are for the 9.3 release. If you require earlier versions, additional links can be found in [R Server 9.1](#). Recall that Python/Anaconda can only be added to a SQL Server Machine Learning Services instance. Pre-trained models exist for both Python and R; the .cab provides models in the languages you're using.

Feature	Download
R	SRO_3.4.3.0_1033.cab ↗
Python	SPO_9.3.0.0_1033.cab ↗
Pre-trained models	MLM_9.3.0.0_1033.cab ↗

3. Transfer .zip and .cab files to the target server.
4. On the server, type `%temp%` in the Run command to get the physical location of the temp directory. The physical path varies by machine, but it's usually `C:\Users\
<your-user-name>\AppData\Local\Temp`.
5. Place the .cab files in the `%temp%` folder.
6. Unzip the Installer.
7. Run ServerSetup.exe and follow the on-screen prompts to complete the installation.

Command-line operations

 Tip

Can't find SqlBindR? You probably have not run Setup. SqlBindR is available only after running Machine Learning Server Setup.

1. Open a command prompt as administrator and navigate to the folder containing `sqlbindr.exe`. The default location is `C:\Program Files\Microsoft\MLServer\Setup`
2. Type the following command to view a list of available instances: `SqlBindR.exe /list`

Make a note of the full instance name as listed. For example, the instance name might be `MSSQL14.MSSQLSERVER` for a default instance, or something like `SERVERNAME.MYNAMEINSTANCE`.

3. Run `SqlBindR.exe` command with the `/bind` argument. Specify the name of the instance to upgrade, using the instance name that was returned in the previous step.

For example, to upgrade the default instance, type: `SqlBindR.exe /bind MSSQL14.MSSQLSERVER`

4. When the upgrade has completed, restart the Launchpad service associated with any instance that has been modified.

Revert or unbind an instance

You can restore a bound instance to an initial installation of the Python and R components, established by SQL Server Setup. There are three parts to reverting back to the SQL Server servicing.

- [Step 1: Unbind from Microsoft Machine Learning Server](#)
- [Step 2: Restore the instance to original status](#)
- [Step 3: Reinstall any packages you added to the installation](#)

Step 1: Unbind

You have two options for rolling back the binding: re-rerun setup or use `SqlBindR` command-line utility.

Unbind using Setup

1. Locate the installer for Machine Learning Server. If you have removed the installer, you may need to download it again, or copy it from another computer.

2. Be sure to run the installer on the computer that has the instance you want to unbind.
3. The installer identifies local instances that are candidates for unbinding.
4. Deselect the check box next to the instance that you want to revert to the original configuration.
5. Accept all licensing agreements.
6. Select **Finish**. The process takes a while.

Unbind using the command line

1. Open a command prompt and navigate to the folder that contains **sqlbindr.exe**, as described in the previous section.
2. Run the **SqlBindR.exe** command with the */unbind* argument, and specify the instance.

For example, the following command reverts the default instance:

```
SqlBindR.exe /unbind MSSQL14.MSSQLSERVER
```

Step 2: Repair the SQL Server instance

Run SQL Server Setup to repair the database engine instance having the Python and R features. Pre-existing updates are preserved. The next step applies if an update was missed for the servicing updates to Python and R packages.

Alternate solution: Fully uninstall and reinstall the database engine instance, and then apply all service updates.

Step 3: Add any third-party packages

You might have added other open-source or third-party packages to your package library. Since reversing the binding switches the location of the default package library, you must reinstall the packages to the library that Python and R are now using. For more information, see [R package information](#) and [installation](#), and [Python package information](#) and [installation](#).

SqlBindR.exe command syntax

Usage

```
sqlbindr [/list] [/bind <SQL_instance_ID>] [/unbind <SQL_instance_ID>]
```

Parameters

Name	Description
<i>list</i>	Displays a list of all SQL Server instance IDs on the current computer
<i>bind</i>	Upgrades the specified SQL Server instance to the latest version of R Server and ensures the instance automatically gets future upgrades of R Server
<i>unbind</i>	Uninstalls the latest version of R Server from the specified SQL Server instance and prevents future R Server upgrades from affecting the instance

Binding errors

Machine Learning Server Installer and SqlBindR both return the following error codes and messages.

Error code	Message	Details
Bind error 0	Ok (success)	Binding passed with no errors.
Bind error 1	Invalid arguments	Syntax error.
Bind error 2	Invalid action	Syntax error.
Bind error 3	Invalid instance	An instance exists, but isn't valid for binding.
Bind error 4	Not bindable	
Bind error 5	Already bound	You ran the <i>bind</i> command, but the specified instance is already bound.

Error code	Message	Details
Bind error 6	Bind failed	An error occurred while unbinding the instance. This error can occur if you run the Machine Learning Server installer without selecting any features. Binding requires that you select both an MSSQL instance and Python and R, assuming the instance is SQL Server 2017. This error also occurs if SqlBindR couldn't write to the Program Files folder. Open sessions or handles to SQL Server will cause this error to occur. If you get this error, reboot the computer and redo the binding steps before starting any new sessions.
Bind error 7	Not bound	The database engine instance has R Services or SQL Server Machine Learning Services. The instance isn't bound to Microsoft Machine Learning Server.
Bind error 8	Unbind failed	An error occurred while unbinding the instance.
Bind error 9	No instances found	No database engine instances were found on this computer.

Known issues

This section lists known issues specific to use of the SqlBindR.exe utility, or to upgrades of Machine Learning Server that might affect SQL Server instances.

Restoring packages that were previously installed

SqlBindR.exe fails to restore original packages or R components with upgrade to Microsoft R Server 9.0.1. Use SQL Server repair on instance and apply all service releases. Restart instance.

Later version of SqlBindR automatically restores the original R features, eliminating the need for reinstallation of R components or repatch the server. However, you must install any R package updates that might have been added after the initial installation.

Use R commands to synchronize installed packages to the file system using records in the database. For more information, see [R package management for SQL Server](#).

Problems with overwritten sqlbinr.ini file in SQL Server

Scenario: This issue occurs when binding Machine Learning Server 9.4.7 to SQL Server 2017. When Python is updated and bound or when you update to a new CU, it doesn't

understand that Python is bound, and overwrites files. There isn't a known issue with R.

As a workaround, create a `sqlbindr.ini` file in the `PYTHON_SERVICES` directory that isn't empty. The contents doesn't impact how the file functions.

Create a `sqlbindr.ini` file, containing **9.4.7.82**, save to this location:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES
```

Problems with multiple upgrades from SQL Server

Scenario: Previously upgraded instance of SQL Server 2016 R Services to 9.0.1. Executed the new installer for Microsoft R Server 9.1.0. The installer displays a list of all valid instances. By default installer selects previously bound instances. If you continue, the previously bound instances are unbound. The result is the earlier 9.0.1 installation is removed and any related packages, but the new version of Microsoft R Server (9.1.0) isn't installed.

As a workaround, you can modify the existing R Server installation as follows:

1. In Control Panel, open **Add or Remove Programs**.
2. Locate Microsoft R Server, and select **Change/Modify**.
3. When the installer starts, select the instances you want to bind to 9.1.0.

Microsoft Machine Learning Server 9.2.1 and 9.3 don't have this issue.

Binding or unbinding leaves multiple temporary folders

Remove temporary folders after installation is complete.

ⓘ Note

Be sure to wait until installation is complete. It can take a long time to remove R libraries associated with one version and then add the new R libraries. When the operation completes, temporary folders are removed.

See also

- [Change the default R or Python language runtime version](#)
- [Install Machine Learning Server for Windows \(Internet connected\)](#)
- [Install Machine Learning Server for Windows \(offline\)](#)

- [Known issues in Machine Learning Server](#)
- [Feature announcements from previous release of R Server](#)
- [Deprecated, no longer supported, or changed features](#)

Run Python and R scripts in Azure Data Studio notebooks with SQL Server Machine Learning Services

Article • 02/28/2023

Applies to:  SQL Server 2017 (14.x) and later

Learn how to run Python and R scripts in [Azure Data Studio](#) notebooks with [SQL Server Machine Learning Services](#). Azure Data Studio is a cross-platform database tool.

Prerequisites

- [Download and install Azure Data Studio](#) on your workstation computer. Azure Data Studio is cross-platform, and runs on Windows, macOS, and Linux.
- A server with SQL Server Machine Learning Services installed and enabled. You can use Machine Learning Services on Windows, Linux, or Big Data Clusters:
 - [Install SQL Server Machine Learning Services on Windows](#).
 - [Install SQL Server Machine Learning Services on Linux](#).
 - [Run Python and R scripts with Machine Learning Services on SQL Server Big Data Clusters](#).

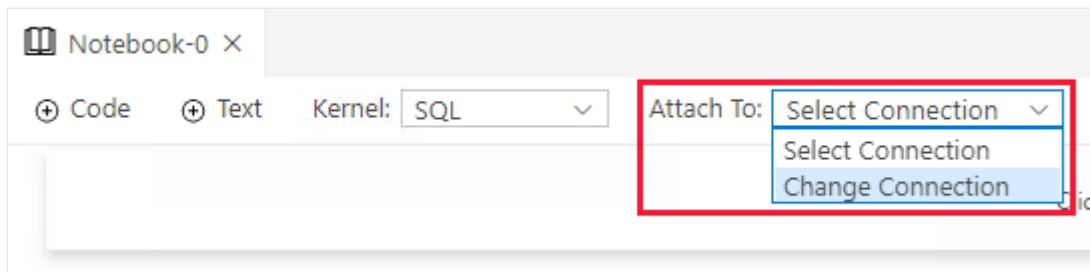
Create a SQL notebook

Important

Machine Learning Services runs as part of SQL Server. Therefore, you need to use a SQL kernel and not a Python kernel.

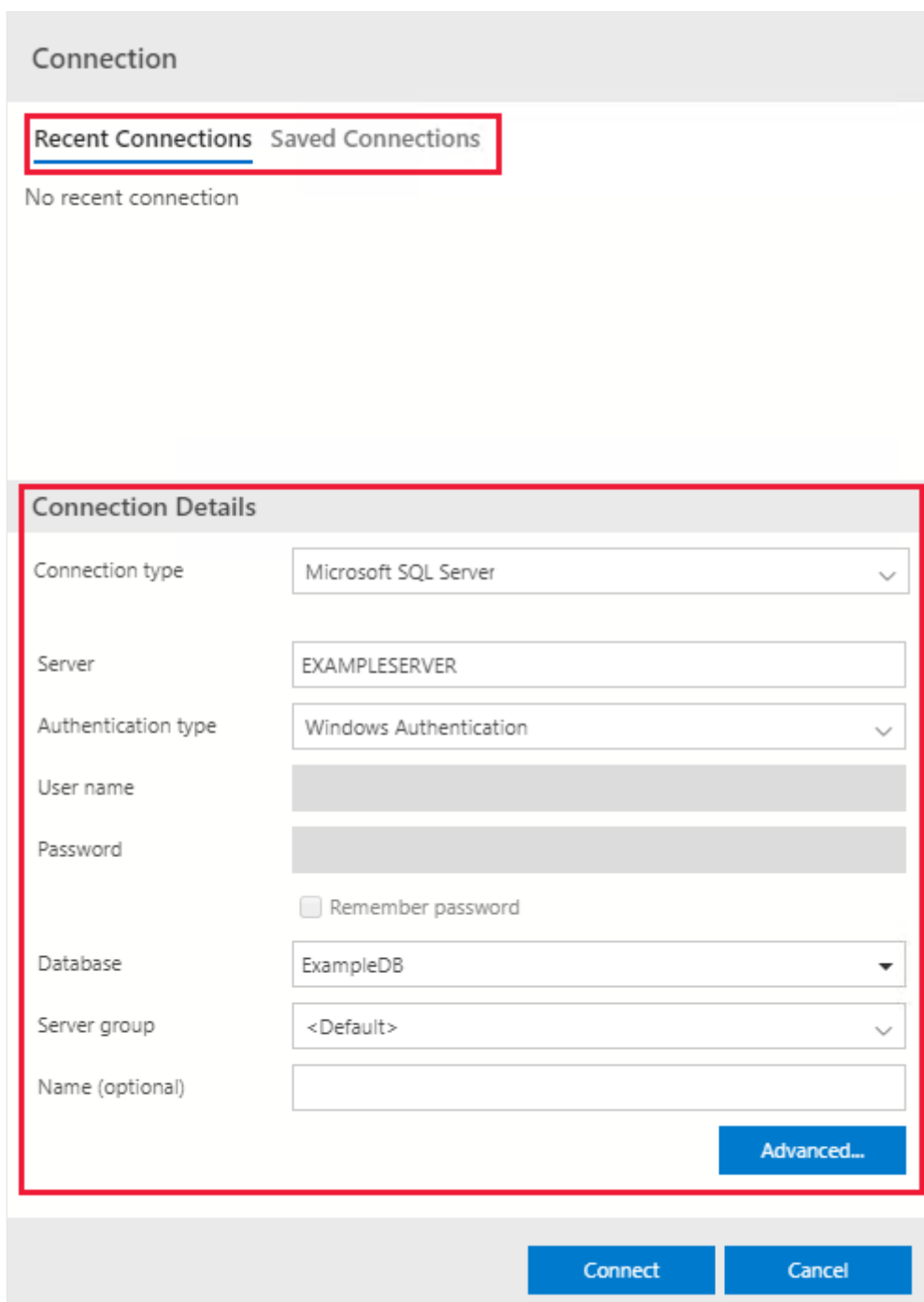
You can use Machine Learning Services in Azure Data Studio with a SQL notebook. To create a new notebook, follow these steps:

1. Click **File** and **New Notebook** to create a new notebook. The notebook will by default use the **SQL kernel**.
2. Click **Attach To** and **Change Connection**.



3. Connect to an existing or new SQL Server. You can either:

- a. Choose an existing connection under **Recent Connections** or **Saved Connections**.
- b. Create a new connection under **Connection Details**. Fill out the connection details to your SQL Server and database.



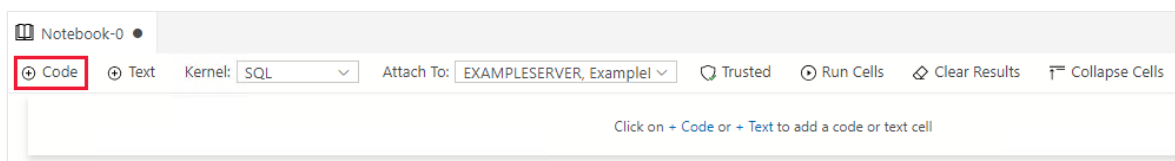
Run Python or R scripts

SQL Notebooks consist of code and text cells. Code cells are used to run Python or R scripts via the stored procedure `sp_execute_external_scripts`. Text cells can be used to document your code in the notebook.

Run a Python script

Follow these steps to run a Python script:

1. Click **+ Code** to add a code cell.

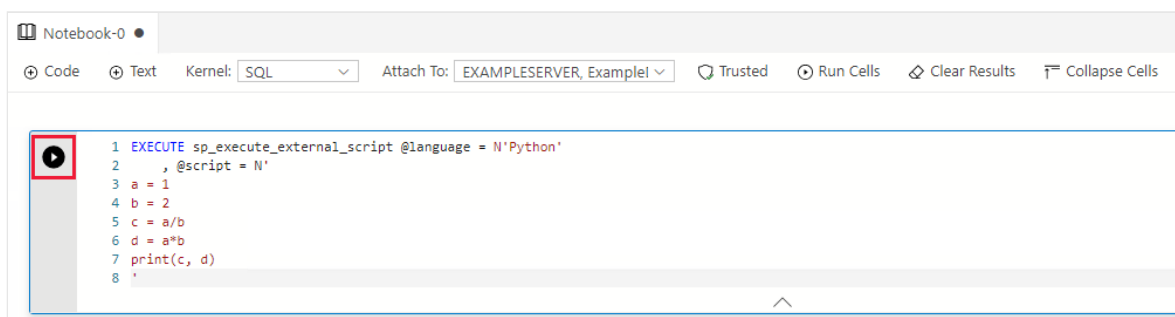


2. Enter the following script in the code cell:

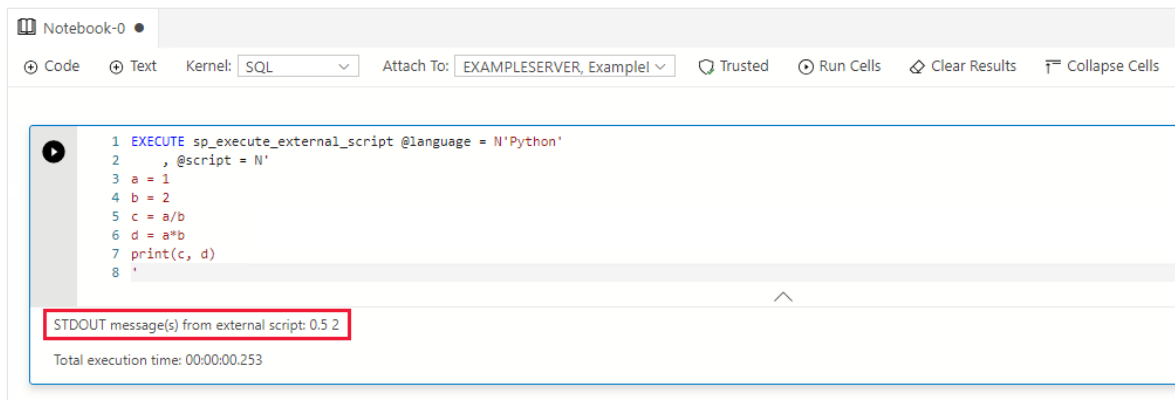
```
SQL

EXECUTE sp_execute_external_script @language = N'Python'
    , @script = N'
a = 1
b = 2
c = a/b
d = a*b
print(c, d)
'
```

3. Click **Run cell** (the round black arrow) or press **F5** to run the single cell.



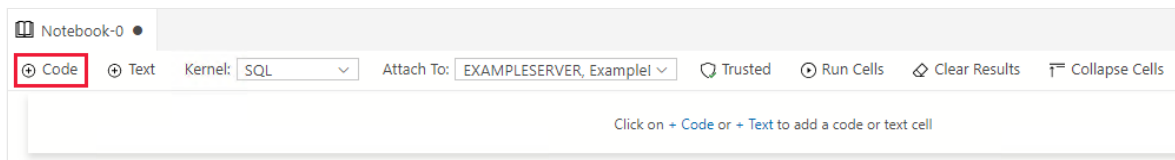
4. The result will be shown under the code cell.



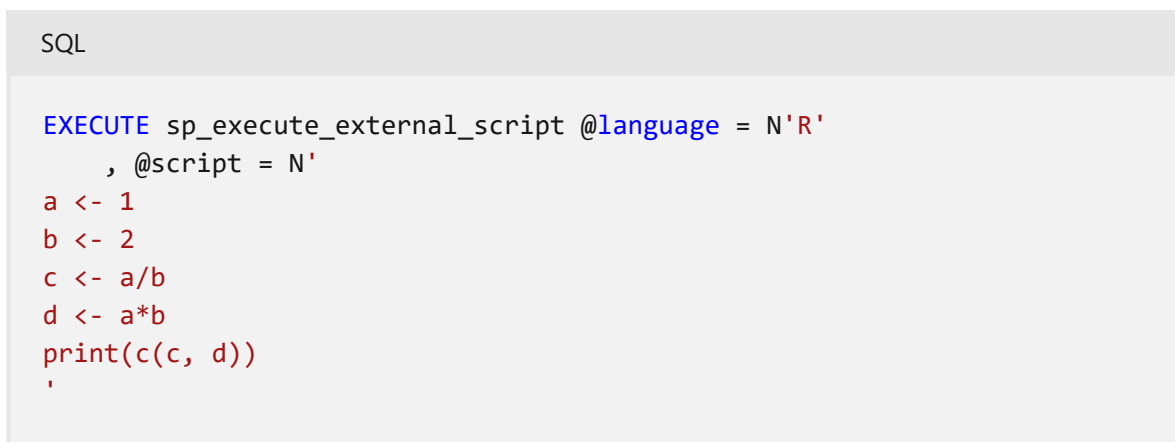
Run an R script

Follow these steps to run an R script:

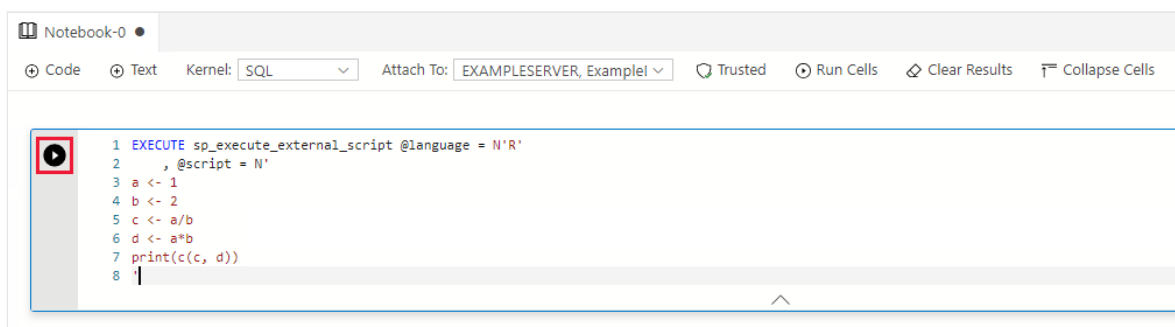
1. Click + **Code** to add a code cell.



2. Enter the following script in the code cell:



3. Click **Run cell** (the round black arrow) or press **F5** to run the single cell.



4. The result will be shown under the code cell.

```
1 EXECUTE sp_execute_external_script @language = N'R'
2     , @script = N'
3 a <- 1
4 b <- 2
5 c <- a/b
6 d <- a*b
7 print(c(c, d))
8 '
```

STDOUT message(s) from external script: [1] 0.5 2.0

Total execution time: 00:00:00.174

Next steps

- [How to use notebooks in Azure Data Studio](#)
- [Create and run a SQL Server notebook](#)
- [Quickstart: Run simple Python scripts with SQL Server Machine Learning Services](#)
- [Quickstart: Run simple R scripts with SQL Server Machine Learning Services](#)

Set up a data science client for Python development on SQL Server Machine Learning Services

Article • 03/03/2023

Applies to: ✓ SQL Server 2016 (13.x), ✓ SQL Server 2017 (14.x), and ✓ SQL Server 2019 (15.x), ✓ SQL Server 2019 (15.x) - Linux

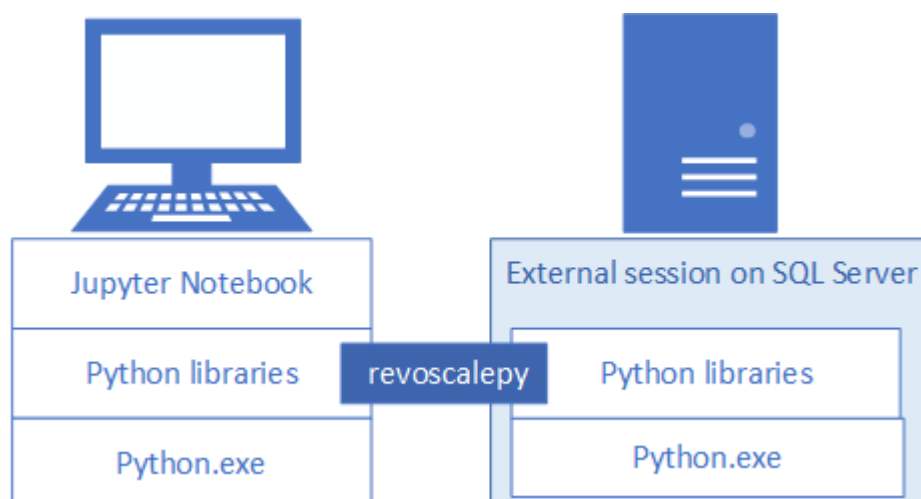
Python integration is available in SQL Server 2017 and later, when you include the Python option in a [Machine Learning Services \(In-Database\) installation](#).

ⓘ Note

Currently this article applies to SQL Server 2016 (13.x), SQL Server 2017 (14.x), SQL Server 2019 (15.x), and SQL Server 2019 (15.x) for Linux only.

To develop and deploy Python solutions for SQL Server, install Microsoft's [revoscalepy](#) and other Python libraries your development workstation. The revoscalepy library, which is also on the remote SQL Server instance, coordinates computing requests between both systems.

In this article, learn how to configure a Python development workstation so that you can interact with a remote SQL Server enabled for machine learning and Python integration. After completing the steps in this article, you will have the same Python libraries as those on SQL Server. You will also know how to push computations from a local Python session to a remote Python session on SQL Server.



To validate the installation, you can use built-in Jupyter Notebooks as described in this article, or [link the libraries](#) to PyCharm or any another IDE that you normally use.

💡 Tip

For a video demonstration of these exercises, see [Run R and Python remotely in SQL Server from Jupyter Notebooks](#) [↗].

Commonly used tools

Whether you are a Python developer new to SQL, or a SQL developer new to Python and in-database analytics, you will need both a Python development tool and a T-SQL query editor such as [SQL Server Management Studio \(SSMS\)](#) to exercise all of the capabilities of in-database analytics.

For Python development, you can use Jupyter Notebooks, which come bundled in the Anaconda distribution installed by SQL Server. This article explains how to start Jupyter Notebooks so that you can run Python code locally and remotely on SQL Server.

SSMS is a separate download, useful for creating and running stored procedures on SQL Server, including those containing Python code. Almost any Python code that you write in Jupyter Notebooks can be embedded in a stored procedure. You can step through other quickstarts to learn about [SSMS and embedded Python](#).

1 - Install Python packages

Local workstations must have the same Python package versions as those on SQL Server, including the base [Anaconda 4.2.0](#) [↗] with [Python 3.5.2 distribution](#) [↗], and Microsoft-specific packages.

An installation script adds three Microsoft-specific libraries to the Python client. The script installs:

- [revoscalepy](#), used for defining data source objects and the compute context.
- [microsoftml](#) providing machine learning algorithms.
- [azureml](#) which applies to operationalization tasks associated with a standalone server context and might be of limited use for in-database analytics.

1. Download an installation script.

- <https://aka.ms/mls-py> installs version 9.2.1 of the Microsoft Python packages. This version corresponds to a default SQL Server instance.
- <https://aka.ms/mls93-py> installs version 9.3 of the Microsoft Python packages.

2. Open a PowerShell window with elevated administrator permissions (right-click **Run as administrator**).

3. Go to the folder in which you downloaded the installer and run the script. Add the `-InstallFolder` command-line argument to specify a folder location for the libraries. For example:

```
Python

cd {{download-directory}}
.\Install-PyForMLS.ps1 -InstallFolder "C:\path-to-python-for-mls"
```

If you omit the install folder, the default is `%ProgramFiles%\Microsoft\PyForMLS`.

Installation takes some time to complete. You can monitor progress in the PowerShell window. When setup is finished, you have a complete set of packages.

Tip

We recommend the [Python for Windows FAQ](#) for general purpose information on running Python programs on Windows.

2 - Locate executables

Still in PowerShell, list the contents of the installation folder to confirm that `Python.exe`, scripts, and other packages are installed.

1. Enter `cd \` to go to the root drive, and then enter the path you specified for `-InstallFolder` in the previous step. If you omitted this parameter during installation, the default is `cd %ProgramFiles%\Microsoft\PyForMLS`.
2. Enter `dir *.exe` to list the executables. You should see `python.exe`, `pythonw.exe`, and `uninstall-anaconda.exe`.

```
PS C:\mspythonlibs> dir *.exe

Directory: C:\mspythonlibs

Mode                LastWriteTime         Length Name
----                -
-a----             7/5/2016   9:41 AM         34816 python.exe
-a----             7/5/2016   9:41 AM         34816 pythonw.exe
-a----             1/19/2017   2:47 PM        699017 Uninstall-Anaconda.exe
```

On systems having multiple versions of Python, remember to use this particular Python.exe if you want to load **revoscalepy** and other Microsoft packages.

ⓘ Note

The installation script does not modify the PATH environment variable on your computer, which means that the new python interpreter and modules you just installed are not automatically available to other tools you might have. For help on linking the Python interpreter and libraries to tools, see **Install an IDE**.

3 - Open Jupyter Notebooks

Anaconda includes Jupyter Notebooks. As a next step, create a notebook and run some Python code containing the libraries you just installed.

1. At the PowerShell prompt, still in the `%ProgramFiles%\Microsoft\PyForMLS` directory, open Jupyter Notebooks from the Scripts folder:

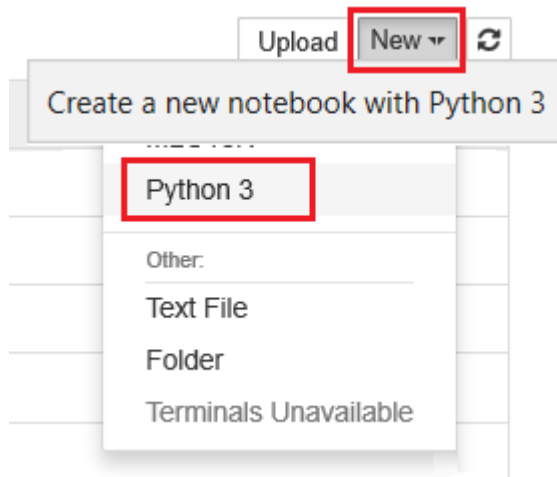
```
PowerShell

.\Scripts\jupyter-notebook
```

A notebook should open in your default browser at `https://localhost:8889/tree`.

Another way to start is double-click `jupyter-notebook.exe`.

2. Select **New** and then select **Python 3**.



3. Enter `import revoscalepy` and run the command to load one of the Microsoft-specific libraries.
4. Enter and run `print(revoscalepy.__version__)` to return the version information. You should see 9.2.1 or 9.3.0. You can use either of these versions with [revoscalepy on the server](#).
5. Enter a more complex series of statements. This example generates summary statistics using `rx_summary` over a local data set. Other functions get the location of the sample data and create a data source object for a local `.xdf` file.

```
Python

import os
from revoscalepy import rx_summary
from revoscalepy import RxDxfData
from revoscalepy import RxOptions
sample_data_path = RxOptions.get_option("sampleDataDir")
print(sample_data_path)
ds = RxDxfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))
summary = rx_summary("ArrDelay+DayOfWeek", ds)
print(summary)
```

The following screenshot shows the input and a portion of the output, trimmed for brevity.

```
In [2]: import os
from revoscalepy import rx_summary
from revoscalepy import RxXdfData
from revoscalepy import RxOptions
sample_data_path = RxOptions.get_option("sampleDataDir")
print(sample_data_path)
ds = RxXdfData(os.path.join(sample_data_path, "AirlineDemoSmall.xdf"))
summary = rx_summary("ArrDelay+DayOfWeek", ds)
print(summary)

C:\mls-python-client\lib\site-packages\revoscalepy\data\sample_data
Rows Read: 200000, Total Rows Processed: 200000, Total Chunk Time: 0.008 seconds
Rows Read: 200000, Total Rows Processed: 400000, Total Chunk Time: 0.008 seconds
Rows Read: 200000, Total Rows Processed: 600000, Total Chunk Time: 0.005 seconds
Computation time: 0.035 seconds.
Call:
rx_summary(formula = 'ArrDelay+DayOfWeek', data = <revoscalepy.datasource.RxXdfData.RxXdfData object at
0x000001AD7C0E3828>, by_group_out_file = None, summary_stats = ['Mean', 'StdDev', 'Min', 'Max', 'ValidObs',
'MissingObs'], by_term = True, pweights = None, fweights = None, row_selection = None, transforms = None,
transform_objects = None, transform_function = None, transform_variables = None, transform_packages = None,
transform_environment = None, overwrite = False, use_sparse_cube = False, remove_zero_counts = False,
blocks_per_read = 1, rows_per_block = 100000, report_progress = None, verbose = 0, compute_context =
<revoscalepy.computecontext.RxLocalSeq.RxLocalSeq object at 0x000001AD7C2C54A8>)

Summary Statistics Results for: ArrDelay+DayOfWeek
File name: C:\mls-python-client\lib\site-packages\revoscalepy\data\sample_data\AirlineDemoSmall.xdf
Number of valid observations: 600000.0
```

4 - Get SQL permissions

To connect to an instance of SQL Server to run scripts and upload data, you must have a valid login on the database server. You can use either a SQL login or integrated Windows authentication. We generally recommend that you use Windows integrated authentication, but using the SQL login is simpler for some scenarios, particularly when your script contains connection strings to external data.

At a minimum, the account used to run code must have permission to read from the databases you are working with, plus the special permission EXECUTE ANY EXTERNAL SCRIPT. Most developers also require permissions to create stored procedures, and to write data into tables containing training data or scored data.

Ask the database administrator to [configure the following permissions for your account](#), in the database where you use Python:

- **EXECUTE ANY EXTERNAL SCRIPT** to run Python on the server.
- **db_datareader** privileges to run the queries used for training the model.
- **db_datawriter** to write training data or scored data.
- **db_owner** to create objects such as stored procedures, tables, functions. You also need **db_owner** to create sample and test databases.

If your code requires packages that are not installed by default with SQL Server, arrange with the database administrator to have the packages installed with the instance. SQL Server is a secured environment and there are restrictions on where packages can be installed. Ad hoc installation of packages as part of your code is not recommended, even if you have rights. Also, always carefully consider the security implications before installing new packages in the server library.

5 - Create test data

If you have permissions to create a database on the remote server, you can run the following code to create the Iris demo database used for the remaining steps in this article.

5-1 - Create the irissql database remotely

Python

```
import pyodbc

# creating a new db to load Iris sample in
new_db_name = "irissql"
connection_string = "Driver=SQL Server;Server=localhost;Database=
{0};Trusted_Connection=Yes;"
                    # you can also swap Trusted_Connection for UID={your
                    # username};PWD={your password}
cnxn = pyodbc.connect(connection_string.format("master"), autocommit=True)
cnxn.cursor().execute("IF EXISTS(SELECT * FROM sys.databases WHERE [name] =
'{0}') DROP DATABASE {0}".format(new_db_name))
cnxn.cursor().execute("CREATE DATABASE " + new_db_name)
cnxn.close()

print("Database created")
```

5-2 - Import Iris sample from SkLearn

Python

```
from sklearn import datasets
import pandas as pd

# SkLearn has the Iris sample dataset built in to the package
iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

5-3 - Use Revoscalepy APIs to create a table and load the Iris data

Python

```
from revoscalepy import RxSqlServerData, rx_data_step

# Example of using RX APIs to load data into SQL table. You can also do this
```

```

with pyodbc
table_ref =
RxSqlServerData(connection_string=connection_string.format(new_db_name),
table="iris_data")
rx_data_step(input_data = df, output_file = table_ref, overwrite = True)

print("New Table Created: Iris")
print("Sklearn Iris sample loaded into Iris table")

```

6 - Test remote connection

Before trying this next step, make sure you have permissions on the SQL Server instance and a connection string to the [Iris sample database](#). If the database doesn't exist and you have sufficient permissions, you can [create a database using these inline instructions](#).

Replace the connection string with valid values. The sample code uses `"Driver=SQL Server;Server=localhost;Database=irissql;Trusted_Connection=Yes;"` but your code should specify a remote server, possibly with an instance name, and a credential option that maps to a database login.

6-1 Define a function

The following code defines a function that you will send to SQL Server in a later step. When executed, it uses data and libraries (revoscalepy, pandas, matplotlib) on the remote server to create scatter plots of the iris data set. It returns the bytestream of the .png back to Jupyter Notebooks to render in the browser.

Python

```

def send_this_func_to_sql():
    from revoscalepy import RxSqlServerData, rx_import
    from pandas.tools.plotting import scatter_matrix
    import matplotlib.pyplot as plt
    import io

    # remember the scope of the variables in this func are within our SQL
    Server Python Runtime
    connection_string = "Driver=SQL
Server;Server=localhost;Database=irissql;Trusted_Connection=Yes;"

    # specify a query and load into pandas dataframe df
    sql_query = RxSqlServerData(connection_string=connection_string,
sql_query = "select * from iris_data")
    df = rx_import(sql_query)

```

```

scatter_matrix(df)

# return bytestream of image created by scatter_matrix
buf = io.BytesIO()
plt.savefig(buf, format="png")
buf.seek(0)

return buf.getvalue()

```

6-2 Send the function to SQL Server

In this example, create the remote compute context and then send the execution of the function to SQL Server with `rx_exec`. The `rx_exec` function is useful because it accepts a compute context as an argument. Any function that you want to execute remotely must have a compute context argument. Some functions, such as `rx_lin_mod` support this argument directly. For operations that don't, you can use `rx_exec` to deliver your code in a remote compute context.

In this example, no raw data had to be transferred from SQL Server to the Jupyter Notebook. All computations occur within the Iris database and only the image file is returned to the client.

Python

```

from IPython import display
import matplotlib.pyplot as plt
from revoscalepy import RxInSqlServer, rx_exec

# create a remote compute context with connection to SQL Server
sql_compute_context =
RxInSqlServer(connection_string=connection_string.format(new_db_name))

# use rx_exec to send the function execution to SQL Server
image = rx_exec(send_this_func_to_sql, compute_context=sql_compute_context)
[0]

# only an image was returned to my jupyter client. All data remained secure
and was manipulated in my db.
display.Image(data=image)

```

The following screenshot shows the input and scatter plot output.

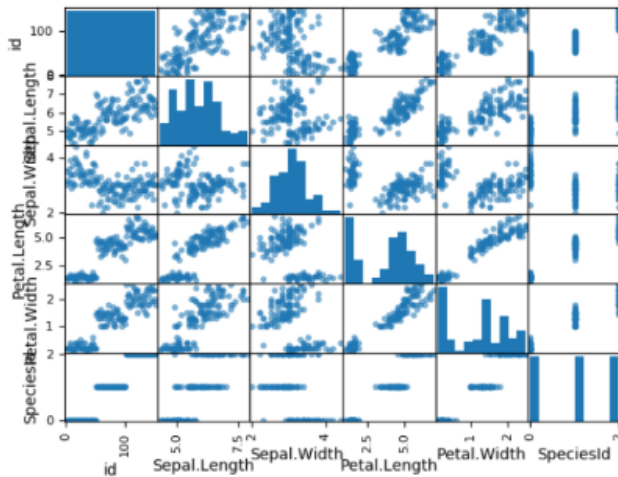

```
In [15]: from IPython import display
import matplotlib.pyplot as plt
from revoscalepy import RxInSqlServer, rx_exec

# create a remote compute context with connection to SQL Server
sql_compute_context = RxInSqlServer(connection_string=connection_string.format(new_db_name))

# use rx_exec to send the function execution to SQL Server
image = rx_exec(send_this_func_to_sql, compute_context=sql_compute_context)[0]

# only an image was returned to my jupyter client. ALL data remained secure and was manipulated in my db.
display.Image(data=image)
```

Out[15]:



7 - Start Python from tools

Because developers frequently work with multiple versions of Python, setup does not add Python to your PATH. To use the Python executable and libraries installed by setup, link your IDE to **Python.exe** at the path that also provides **revoscalepy** and **microsoftml**.

Command line

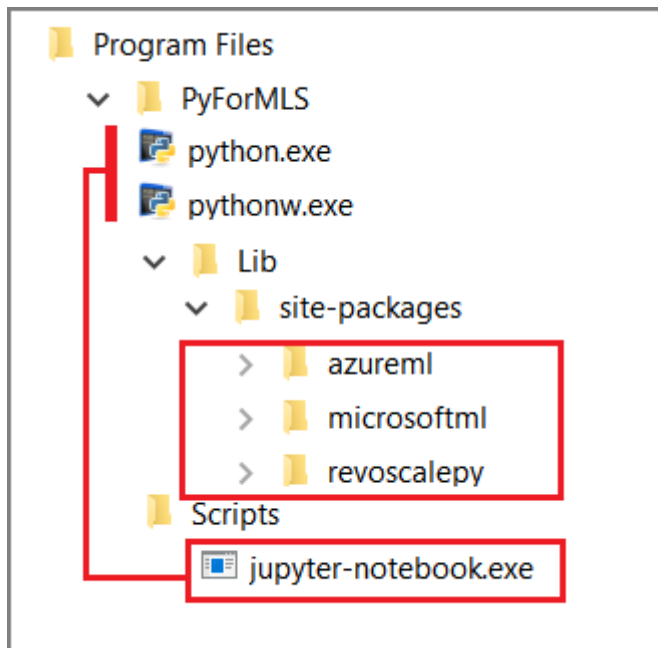
When you run **Python.exe** from `%ProgramFiles%\Microsoft\PyForMLS` (or whatever location you specified for the Python client library installation), you have access to the full Anaconda distribution plus the Microsoft Python modules, **revoscalepy** and **microsoftml**.

1. Go to `%ProgramFiles%\Microsoft\PyForMLS` and execute **Python.exe**.
2. Open interactive help: `help()`.
3. Type the name of a module at the help prompt: `help> revoscalepy`. Help returns the name, package contents, version, and file location.
4. Return version and package information at the `help>` prompt: `revoscalepy`. Press Enter a few times to exit help.
5. Import a module: `import revoscalepy`.

Jupyter Notebooks

This article uses built-in Jupyter Notebooks to demonstrate function calls to **revoscalepy**. If you are new to this tool, the following screenshot illustrates how the pieces fit together and why it all "just works".

The parent folder `%ProgramFiles%\Microsoft\PyForMLS` contains Anaconda plus the Microsoft packages. Jupyter Notebooks is included in Anaconda, under the Scripts folder, and the Python executables are auto-registered with Jupyter Notebooks. Packages found under site-packages can be imported into a notebook, including the three Microsoft packages used for data science and machine learning.



If you are using another IDE, you will need to link the Python executables and function libraries to your tool. The following sections provide instructions for commonly used tools.

Visual Studio

If you have [Python in Visual Studio](#), use the following configuration options to create a Python environment that includes the Microsoft Python packages.

Configuration setting	value
Prefix path	<code>%ProgramFiles%\Microsoft\PyForMLS</code>
Interpreter path	<code>%ProgramFiles%\Microsoft\PyForMLS\python.exe</code>
Windowed interpreter	<code>%ProgramFiles%\Microsoft\PyForMLS\pythonw.exe</code>

For help with configuring a Python environment, see [Managing Python environments in Visual Studio](#).

PyCharm

In PyCharm, set the interpreter to the Python executable installed.

1. In a new project, in Settings, select **Add Local**.
2. Enter `%ProgramFiles%\Microsoft\PyForML\`.

You can now import `revoscalepy`, `microsoftml`, or `azureml` modules. You can also choose **Tools > Python Console** to open an interactive window.

Next steps

Now that you have tools and a working connection to SQL Server, expand your skills by running through the Python quickstarts using [SQL Server Management Studio \(SSMS\)](#).

Quickstart: Create and run simple Python scripts with SQL Server Machine Learning Services

Set up a data science client for R development on SQL Server

Article • 03/03/2023

Applies to: ✔ SQL Server 2016 (13.x), ✔ SQL Server 2017 (14.x), and ✔ SQL Server 2019 (15.x), ✔ SQL Server 2019 (15.x) - Linux

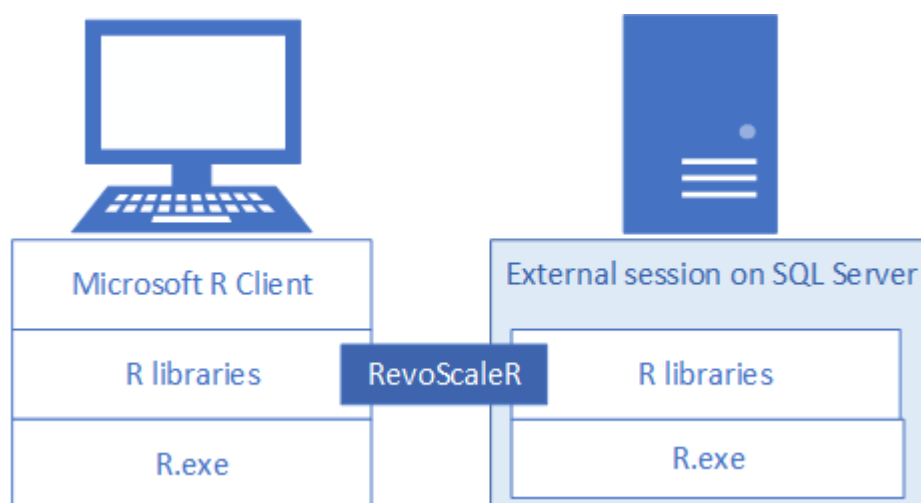
R integration is available in SQL Server 2016 or later when you include the R language option in an [SQL Server 2016 R Services](#) or [SQL Server Machine Learning Services \(In-Database\)](#) installation.

ⓘ Note

Currently this article applies to SQL Server 2016 (13.x), SQL Server 2017 (14.x), SQL Server 2019 (15.x), and SQL Server 2019 (15.x) for Linux only.

To develop and deploy R solutions for SQL Server, install [Microsoft R Client](#) on your development workstation to get [RevoScaleR](#) and other R libraries. The RevoScaleR library, which is also required on the remote SQL Server instance, coordinates computing requests between both systems.

In this article, learn how to configure an R client development workstation so that you can interact with a remote SQL Server enabled for machine learning and R integration. After completing the steps in this article, you will have the same R libraries as those on SQL Server. You will also know how to push computations from a local R session to a remote R session on SQL Server.



To validate the installation, you can use built-in **RGUI** tool as described in this article, or [link the libraries to RStudio or any another IDE](#) that you normally use.

Commonly used tools


Whether you are an R developer new to SQL, or a SQL developer new to R and in-database analytics, you will need both an R development tool and a T-SQL query editor such as [SQL Server Management Studio \(SSMS\)](#) to exercise all of the capabilities of in-database analytics.

For simple R development scenarios, you can use the RGUI executable, bundled in the base R distribution in MRO and SQL Server. This article explains how to use RGUI for both local and remote R sessions. For improved productivity, you should use a full-featured IDE such as [RStudio](#) or [Visual Studio](#).

SSMS is a separate download, useful for creating and running stored procedures on SQL Server, including those containing R code. Almost any R code that you write in a development environment can be embedded in a stored procedure. You can step through other tutorials to learn about [SSMS and embedded R](#).

1 - Install R packages

Microsoft's R packages are available in multiple products and services. On a local workstation, we recommend installing Microsoft R Client. R Client provides [RevoScaleR](#), [MicrosoftML](#), [SQLRUtils](#), and other R packages.

1. [Download Microsoft R Client](#) .
2. In the installation wizard, accept or change default installation path, accept or change the components list, and accept the Microsoft R Client license terms.

When installation is finished, a welcome screen introduces you to the product and documentation.

3. Create an MKL_CBWR system environment variable to ensure consistent output on Intel Math Kernel Library (MKL) calculations.
 - In Control Panel, select **System and Security** > **System** > **Advanced System Settings** > **Environment Variables**.
 - Create a new System variable named **MKL_CBWR**, with a value set to **AUTO**.

2 - Locate executables

Locate and list the contents of the installation folder to confirm that R.exe, RGUI, and other packages are installed.

1. In File Explorer, open the `%ProgramFiles%\Microsoft\R Client\R_SERVER\bin` folder to confirm the location of `R.exe`.
2. Open the x64 subfolder to confirm **RGUI**. You will use this tool in the next step.
3. Open `%ProgramFiles%\Microsoft\R Client\R_SERVER\library` to review the list of packages installed with R Client, including RevoScaleR, MicrosoftML, and others.

3 - Start RGUI

When you install R with SQL Server, you get the same R tools that are standard to any base installation of R, such as RGui, Rterm, and so forth. These tools are lightweight, useful for checking package and library information, running ad hoc commands or script, or stepping through tutorials. You can use these tools to get R version information and confirm connectivity.

1. Open `%ProgramFiles%\Microsoft\R Client\R_SERVER\bin\x64` and double-click **RGui** to start an R session with an R command prompt.

When you start an R session from a Microsoft program folder, several packages, including RevoScaleR, load automatically.

2. Enter `print(Revo.version)` at the command prompt to return RevoScaleR package version information. You should have version 9.2.1 or 9.3.0 for RevoScaleR.
3. Enter `search()` at the R prompt for a list of installed packages.

```
R Console

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Microsoft R Open 3.4.3
The enhanced R distribution from Microsoft
Microsoft packages Copyright (C) 2018 Microsoft

Loading Microsoft R Client packages, version 3.4.3.0097.
Microsoft R Client limits some functions to available memory.
See: https://go.microsoft.com/fwlink/?linkid=799476 for information
about additional features.

Type 'readme()' for release notes, privacy() for privacy policy, or
'RevoLicense()' for licensing information.

Using the Intel MKL for parallel mathematical computing(using 4 cores).
Default CRAN mirror snapshot taken on 2018-01-01.
See: https://mran.microsoft.com/.

> search()
.[1] ".GlobalEnv"           "package:RevoUtilsMath"
 [3] "package:RevoUtils"    "package:RevoMods"
 [5] "package:MicrosoftML" "package:mrsdeploy"
 [7] "package:RevoScaleR"  "package:lattice"
 [9] "package:rpart"       "package:stats"
[11] "package:graphics"    "package:grDevices"
[13] "package:utils"       "package:datasets"
[15] "package:methods"     "Autoloads"
[17] "package:base"
>
```

4 - Get SQL permissions

In R Client, R processing is capped at two threads and in-memory data. For scalable processing using multiple cores and large data sets, you can shift execution (referred to as *compute context*) to the data sets and computational power of a remote SQL Server instance. This is the recommended approach for client integration with a production SQL Server instance, and you will need permissions and connection information to make it work.

To connect to an instance of SQL Server to run scripts and upload data, you must have a valid login on the database server. You can use either a SQL login or integrated Windows authentication. We generally recommend that you use Windows integrated authentication, but using the SQL login is simpler for some scenarios, particularly when your script contains connection strings to external data.

At a minimum, the account used to run code must have permission to read from the databases you are working with, plus the special permission EXECUTE ANY EXTERNAL SCRIPT. Most developers also require permissions to create stored procedures, and to write data into tables containing training data or scored data.

Ask the database administrator to [configure the following permissions for your account](#), in the database where you use R:

- **EXECUTE ANY EXTERNAL SCRIPT** to run R script on the server.
- **db_datareader** privileges to run the queries used for training the model.
- **db_datawriter** to write training data or scored data.
- **db_owner** to create objects such as stored procedures, tables, functions. You also need **db_owner** to create sample and test databases.

If your code requires packages that are not installed by default with SQL Server, arrange with the database administrator to have the packages installed with the instance. SQL Server is a secured environment and there are restrictions on where packages can be installed. For more information, see [Install new R packages on SQL Server](#).

5 - Test connections

As a verification step, use **RGUI** and RevoScaleR to confirm connectivity to the remote server. SQL Server must be enabled for [remote connections](#) and you must have permissions, including a user login and a database to connect to.

The following steps assume the demo database, [NYCTaxi_Sample](#), and Windows authentication.

1. Open **RGUI** on the client workstation. For example, go to `~\Program Files\Microsoft SQL Server\140\R_SERVER\bin\x64` and double-click **RGui.exe** to start it.
2. RevoScaleR loads automatically. Confirm RevoScaleR is operational by running this command: `print(Revo.version)`
3. Enter demo script that executes on the remote server. You must modify the following sample script to include a valid name for a remote SQL Server instance. This session begins as a local session, but the **rxSummary** function executes on the remote SQL Server instance.

```
R
```

```
# Define a connection. Replace server with a valid server name.  
connStr <- "Driver=SQL Server;Server=<your-server->
```



```

name>;Database=NYCTaxi_Sample;Trusted_Connection=true"

# Specify the input data in a SQL query.
sampleQuery <-"SELECT DISTINCT TOP(100) tip_amount FROM
[dbo].nyctaxi_sample ORDER BY tip_amount DESC;"

# Define a remote compute context based on the remote server.
cc <-RxInSqlServer(connectionString=connStr)

# Execute the function using the remote compute context.
rxSummary(formula = ~ ., data = RxSqlServerData(sqlQuery=sampleQuery,
connectionString=connStr), computeContext=cc)

```

Results:

This script connects to a database on the remote server, provides a query, creates a compute context `cc` instruction for remote code execution, then provides the RevoScaleR function `rxSummary` to return a statistical summary of the query results.

```

R

Call:
rxSummary(formula = ~., data = RxSqlServerData(sqlQuery = sampleQuery,
  connectionString = connStr), computeContext = cc)

Summary Statistics Results for: ~.
Data: RxSqlServerData(sqlQuery = sampleQuery, connectionString =
connStr) (RxSqlServerData Data Source)
Number of valid observations: 100

Name      Mean   StdDev  Min Max ValidObs MissingObs
tip_amount 63.245 31.61087 36  180 100      0

```

4. Get and set the compute context. Once you set a compute context, it remains in effect for the duration of the session. If you aren't sure whether computation is local or remote, run the following command to find out. Results that specify a connection string indicate a remote compute context.

```

R

# Return the current compute context.
rxGetCurrentContext()

# Revert to a local compute context.
rxSetComputeContext("local")
rxGetCurrentContext()

# Switch back to remote.

```

```
connStr <- "Driver=SQL Server;Server=<your-server-name>;Database=NYCTaxi_Sample;Trusted_Connection=true"
cc <-RxInSqlServer(connectionString=connStr)
rxSetComputeContext(cc)
rxGetComputeContext()
```

5. Return information about variables in the data source, including name and type.

R

```
rxGetVarInfo(data = inDataSource)
```

Results include 23 variables.

6. Generate a scatter plot to explore whether there are dependencies between two variables.

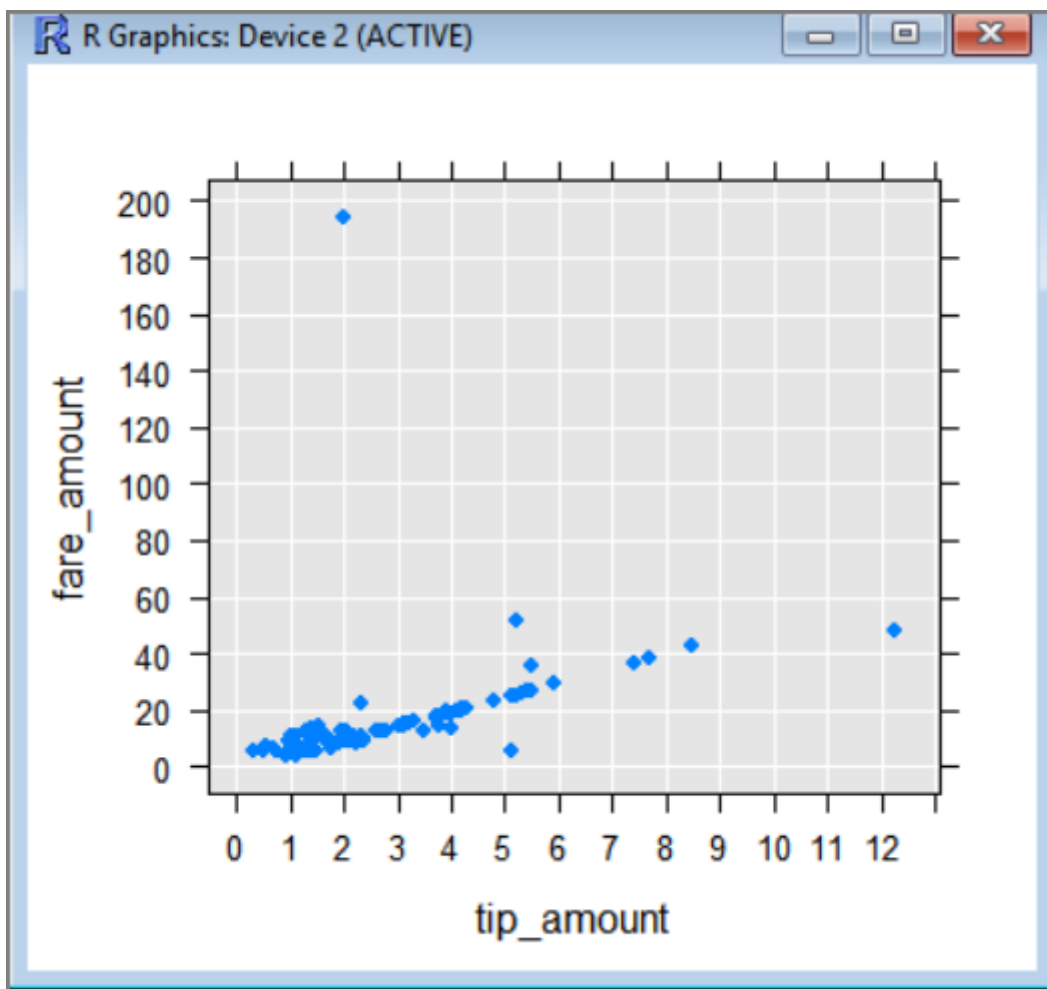
R

```
# Set the connection string. Substitute a valid server name for the
placeholder.
connStr <- "Driver=SQL Server;Server=<your database
name>;Database=NYCTaxi_Sample;Trusted_Connection=true"

# Specify a query on the nyctaxi_sample table.
# For variables on each axis, remove nulls. Use a WHERE clause and <>
to do this.
sampleQuery <- "SELECT DISTINCT TOP 100 * from [dbo].[nyctaxi_sample]
WHERE fare_amount <> '' AND tip_amount <> ''"
cc <-RxInSqlServer(connectionString=connStr)

# Generate a scatter plot.
rxLinePlot(fare_amount ~ tip_amount, data =
RxSqlServerData(sqlQuery=sampleQuery, connectionString=connStr,
computeContext=cc), type="p")
```

The following screenshot shows the input and scatter plot output.



6 - Link tools to R.exe

For sustained and serious development projects, you should install an integrated development environment (IDE). SQL Server tools and the built-in R tools are not equipped for heavy R development. Once you have working code, you can deploy it as a stored procedure for execution on SQL Server.

Point your IDE to the local R libraries: base R, RevoScaleR, and so forth. Running workloads on a remote SQL Server occurs during script execution, when your script invokes a remote compute context on SQL Server, accessing data and operations on that server.

RStudio

When using [RStudio](#), you can configure the environment to use the R libraries and executables that correspond to those on a remote SQL Server.

1. Check R package versions installed on SQL Server. For more information, see [Get R package information](#).

2. Install Microsoft R Client to add RevoScaleR and other R packages, including the base R distribution used by your SQL Server instance. Choose a version at the same level or lower (packages are backward compatible) that provides the same package versions as on the server. To view the package versions installed on the server, see [List all installed R packages](#).
3. In RStudio, [update your R path](#) to point to the R environment providing RevoScaleR, Microsoft R Open, and other Microsoft packages. Look for `%ProgramFiles%\Microsoft\R Client\R_SERVER\bin\x64`.
4. Close and then open RStudio.

When you reopen RStudio, the R executable from R Client is the default R engine.

R Tools for Visual Studio (RTVS)

If you don't already have a preferred IDE for R, we recommend **R Tools for Visual Studio**.

- [Download R Tools for Visual Studio \(RTVS\)](#)
- [Installation instructions](#) - RTVS is available in several versions of Visual Studio.
- [Get started with R Tools for Visual Studio](#)

Connect to SQL Server from RTVS

This example uses Visual Studio 2017 Community Edition, with the data science workload installed.

1. From the **File** menu, select **New** and then select **Project**.
2. The left-hand pane contains a list of preinstalled templates. Select **R**, and select **Project**. In the **Name** box, type `dbtest` and select **OK**.

Visual Studio creates a new project folder and a default script file, `Script.R`.

3. Type `.libPaths()` on the first line of the script file, and then press CTRL + ENTER.

The current R library path should be displayed in the **R Interactive** window.

4. Select the **R Tools** menu and select **Windows** to see a list of other R-specific windows that you can display in your workspace.
 - View help on packages in the current library by pressing CTRL + 3.
 - See R variables in the **Variable Explorer**, by pressing CTRL + 8.


Next steps

Two different tutorials include exercises so that you can practice switching the compute context from local to a remote SQL Server instance.

- [Tutorial: Use RevoScaleR R functions with SQL Server data](#)
- [Data Science End-to-End Walkthrough](#)

Install SQL Server Machine Learning Services with Python and R on an Azure virtual machine

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later

Learn how to install Python and R with [SQL Server Machine Learning Services](#) on a virtual machine in Azure. This eliminates the installation and configuration tasks for Machine Learning Services.

Follow these steps:

1. Provision SQL Server virtual machine in Azure
2. Unblock the firewall
3. Enable ODBC callbacks for remote clients
4. Add network protocols

Provision SQL Server virtual machine in Azure

For step by step instructions, see [How to provision a Windows SQL Server virtual machine in the Azure portal](#).

The [Configure SQL server settings](#) step is where you add Machine Learning Services to your instance.

Unblock the firewall

By default, the firewall on the Azure virtual machine includes a rule that blocks network access for local user accounts.

You must disable this rule to ensure that you can access the SQL Server instance from a remote data science client. Otherwise, your machine learning code cannot execute in compute contexts that use the virtual machine's workspace.

To enable access from remote data science clients:

1. On the virtual machine, open Windows Firewall with Advanced Security.
2. Select **Outbound Rules**

3. Disable the following rule:

```
Block network access for R local user accounts in SQL Server instance  
MSSQLSERVER
```

Enable ODBC callbacks for remote clients

If you expect that clients calling the server will need to issue ODBC queries as part of their machine learning solutions, you must ensure that the Launchpad can make ODBC calls on behalf of the remote client.

To do this, you must allow the SQL worker accounts that are used by Launchpad to log into the instance. For more information, see [Add SQLRUserGroup as a database user](#).

Add network protocols

- Enable Named Pipes



R Services (In-Database) uses the Named Pipes protocol for connections between the client and server computers, and for some internal connections. If Named Pipes is not enabled, you must install and enable it on both the Azure virtual machine, and on any data science clients that connect to the server.

- Enable TCP/IP

TCP/IP is required for loopback connections. If you get the error "DBNETLIB; SQL Server does not exist or access denied", enable TCP/IP on the virtual machine that supports the instance.

Install Machine Learning Server (Standalone) or R Server (Standalone) using SQL Server Setup

Article • 03/17/2023

Applies to:  SQL Server 2016 (13.x),  SQL Server 2017 (14.x), and  SQL Server 2019 (15.x)

Important

The support for Machine Learning Server (previously known as R Server) ended on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

Important

Machine Learning Server (Standalone) is not shipped with SQL Server 2022 (16.x). This article refers to a retired feature of SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x).

SQL Server Setup includes a **shared feature** option for installing a standalone machine learning server that runs outside of SQL Server. It's called **Machine Learning Server (Standalone)** and includes Python and R.

A standalone server as installed by SQL Server Setup supports use cases and scenarios such as the following:

- Remote execution, switching between local and remote sessions in the same console
- Operationalization with web nodes and compute nodes
- Web service deployment: the ability to package R and Python script into web services
- Complete collection of R and Python function libraries

As an independent server decoupled from SQL Server, the R and Python environment is configured, secured, and accessed using the underlying operating system and tools provided in the standalone server, not SQL Server.

As an adjunct to SQL Server, a standalone server is useful if you need to develop high-performance machine learning solutions that can use remote compute contexts to the full range of supported data platforms. You can shift execution from the local server to a remote Machine Learning Server on a Spark cluster or on another SQL Server instance.

Pre-install checklist

If you installed a previous version, such as SQL Server 2016 R Server (Standalone) or Microsoft R Server, uninstall the existing installation before continuing.

As a general rule, we recommend that you treat standalone server and database engine instance-aware installations as mutually exclusive to avoid resource contention, but if you have sufficient resources, there is no prohibition against installing them both on the same physical computer.

You can only have one standalone server on the computer: either SQL Server Machine Learning Server (Standalone) or SQL Server R Server (Standalone). Be sure to uninstall one version before adding a new one.

Get the installation media

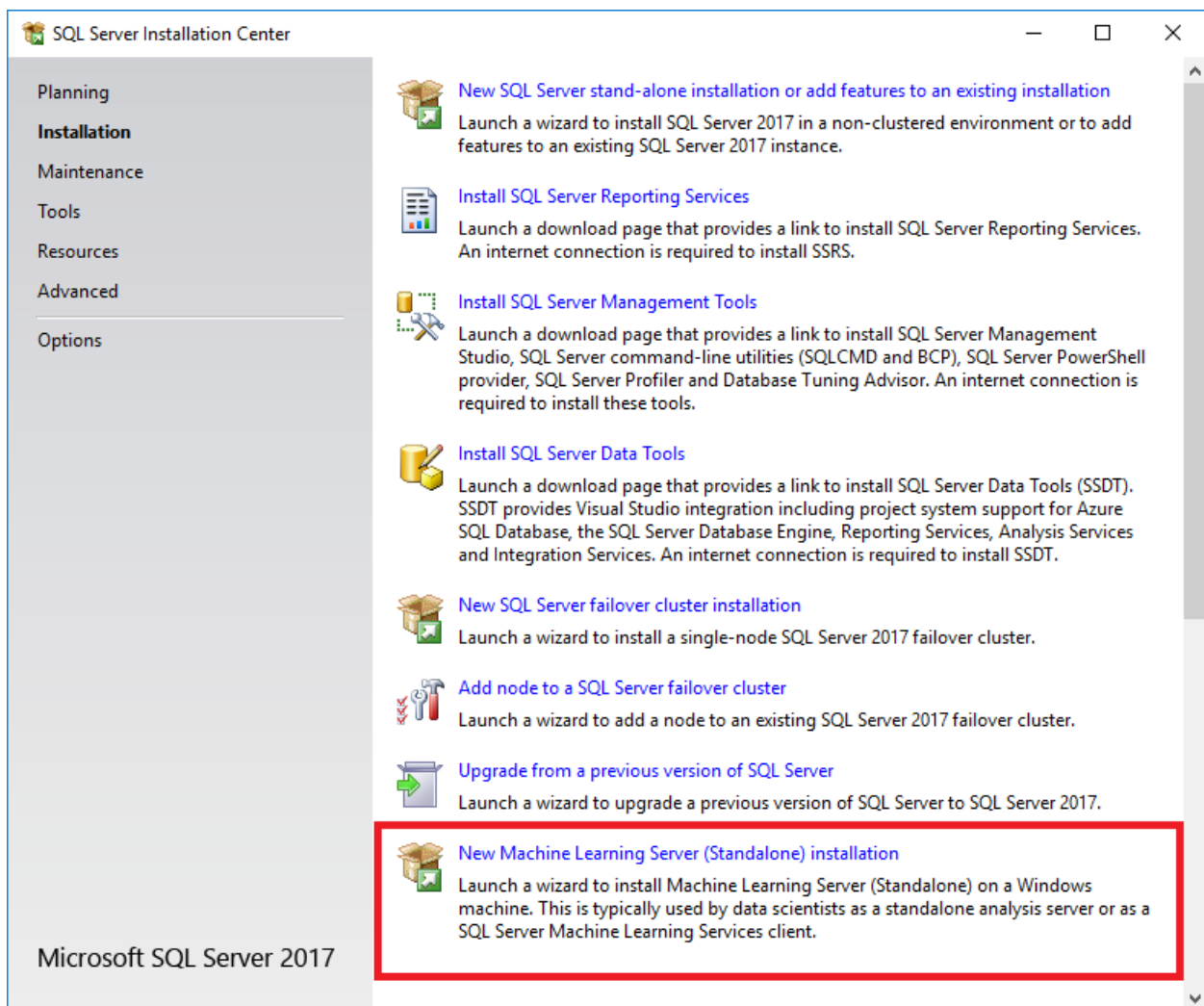
The download location for SQL Server depends on the edition:

- **SQL Server Enterprise, Standard, and Express editions.** These editions are licensed for production use. For the Enterprise and Standard editions, contact your software vendor for the installation media. You can find purchasing information and a directory of Microsoft partners on the [Microsoft purchasing website](#).
- [The latest free edition](#).

Run Setup

For local installations, you must run Setup as an administrator. If you install SQL Server from a remote share, you must use a domain account that has read and execute permissions on the remote share.

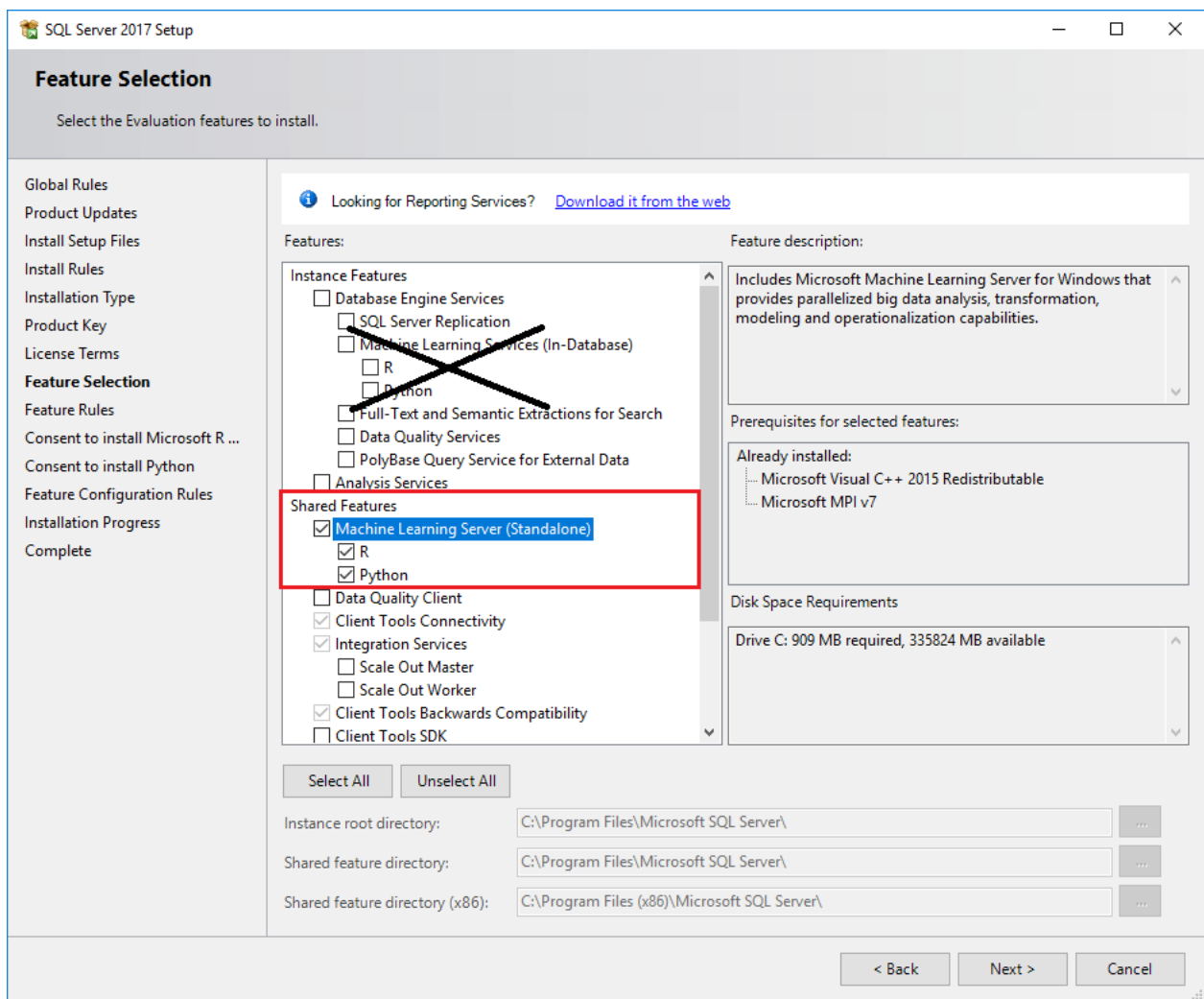
1. Start the installation wizard.
2. Click the **Installation** tab, and select **New Machine Learning Server (Standalone) installation**.



3. After the rules check is complete, accept SQL Server licensing terms, and select a new installation.

4. On the **Feature Selection** page, the following options should already be selected:

- **Microsoft Machine Learning Server (Standalone)**
- **R** and **Python** are both selected by default. You can deselect either language, but we recommend that you install at least one of the supported languages.



All other options should be ignored.

! Note

Avoid installing the **Shared Features** if the computer already has Machine Learning Services installed for SQL Server in-database analytics. This creates duplicate libraries.

Also, whereas R or Python scripts running in SQL Server are managed by SQL Server so as not to conflict with memory used by other database engine services, the standalone machine learning server has no such constraints, and can interfere with other database operations. Finally, remote access via RDP session, which is often used for operationalization, is typically blocked by database administrators.

For these reasons, we generally recommend that you install Machine Learning Server (Standalone) on a separate computer from SQL Server Machine Learning Services.

5. Accept the license terms for downloading and installing base language distributions. When the **Accept** button becomes unavailable, you can click **Next**.

6. On the **Ready to Install** page, verify your selections, and click **Install**.

Set environment variables

For R feature integration only, you should set the **MKL_CBWR** environment variable to [ensure consistent output](#) from Intel Math Kernel Library (MKL) calculations.

1. In Control Panel, click **System and Security > System > Advanced System Settings > Environment Variables**.

2. Create a new User or System variable.

- Set variable name to `MKL_CBWR`
- Set the variable value to `AUTO`

3. Restart the server.

Default installation folders

For R and Python development, it's common to have multiple versions on the same computer. As installed by SQL Server setup, the base distribution is installed in a folder associated with the SQL Server version that you used for setup.

The following table lists the paths for R and Python distributions created by Microsoft installers. For completeness, the table includes paths generated by SQL Server setup as well as the standalone installer for Microsoft Machine Learning Server.

Version	Installation method	Default folder
SQL Server 2019 Machine Learning Server (Standalone)	SQL Server 2019 setup wizard	<code>C:\Program Files\Microsoft SQL Server\150\R_SERVER</code> <code>C:\Program Files\Microsoft SQL Server\150\PYTHON_SERVER</code>
SQL Server 2017 Machine Learning Server (Standalone)	SQL Server 2017 setup wizard	<code>C:\Program Files\Microsoft SQL Server\140\R_SERVER</code> <code>C:\Program Files\Microsoft SQL Server\140\PYTHON_SERVER</code>
Microsoft Machine Learning Server (Standalone)	Windows standalone installer	<code>C:\Program Files\Microsoft\ML Server\R_SERVER</code> <code>C:\Program Files\Microsoft\ML Server\PYTHON_SERVER</code>

Version	Installation method	Default folder
SQL Server Machine Learning Services (In-Database)	SQL Server 2019 setup wizard, with R language option	C:\Program Files\Microsoft SQL Server\MSSQL15. <instance_name>\R_SERVICES C:\Program Files\Microsoft SQL Server\MSSQL15. <instance_name>\PYTHON_SERVICES
SQL Server Machine Learning Services (In-Database)	SQL Server 2017 setup wizard, with R language option	C:\Program Files\Microsoft SQL Server\MSSQL14. <instance_name>\R_SERVICES C:\Program Files\Microsoft SQL Server\MSSQL14. <instance_name>\PYTHON_SERVICES
SQL Server 2016 R Server (Standalone)	SQL Server 2016 setup wizard	C:\Program Files\Microsoft SQL Server\130\R_SERVER
SQL Server 2016 R Services (In-Database)	SQL Server 2016 setup wizard	C:\Program Files\Microsoft SQL Server\MSSQL13. <instance_name>\R_SERVICES

Apply updates

We recommend that you apply the latest cumulative update to both the database engine and machine learning components. Cumulative updates are installed through the Setup program.

On internet-connected devices, you can download a self-extracting executable. Applying an update for the database engine automatically pulls in cumulative updates for existing R and Python features.

On disconnected servers, extra steps are required. You must obtain the cumulative update for the database engine as well as the CAB files for machine learning features. All files must be transferred to the isolated server and applied manually.

1. Start with a baseline instance. You can only apply cumulative updates to existing installations:
 - Machine Learning Server (Standalone) from SQL Server 2019 initial release
 - Machine Learning Server (Standalone) from SQL Server 2017 initial release
 - R Server (Standalone) from SQL Server 2016 initial release, SQL Server 2016 SP 1, or SQL Server 2016 SP 2

2. Close any open R or Python sessions and stop any processes still running on the system.
3. If you enabled operationalization to run as web nodes and compute nodes for web service deployments, back up the **AppSettings.json** file as a precaution. Applying SQL Server 2017 CU13 or later revises this file, so you might want a backup copy to preserve the original version.
4. On an internet connected machine, download the latest cumulative update for your version from the [Latest updates for Microsoft SQL Server](#).
5. Download the latest cumulative update. It is an executable file.
6. On an internet-connected device, double-click the .exe to run Setup and step through the wizard to accept licensing terms, review affected features, and monitor progress until completion.
7. On a server with no internet connectivity:
 - Get corresponding CAB files for R and Python. For download links, see [CAB downloads for cumulative updates on SQL Server in-database analytics instances](#).
 - Transfer all files, the main executable and CAB files, to a folder on the offline computer.
 - Double-click the .exe to run Setup. When installing a cumulative update on a server with no internet connectivity, you are prompted to select the location of the .cab files for R and Python.
8. Post-install, on a server for which you have enabled deployment with web nodes and compute nodes, edit **AppSettings.json**, adding an "MMLResourcePath" entry, directly under "MMLNativePath". For example:

JSON

```
"ScorerParameters": {  
  "MMLNativePath": "C:\\Program Files\\Microsoft SQL  
Server\\140\\R_SERVER\\library\\MicrosoftML\\mxLibs\\x64\\",  
  "MMLResourcePath": "C:\\Program Files\\Microsoft SQL  
Server\\140\\R_SERVER\\library\\MicrosoftML\\mxLibs\\x64\\"  
}
```

9. [Run the admin CLI utility](#) to restart the web and compute nodes. For steps and syntax, see [Monitor, start, and stop web and compute nodes](#).

Development tools

A development IDE is not installed as part of setup. For more information about configuring a development environment, see [Set up R tools](#) and [Set up Python tools](#).

Next steps

R developers can get started with some simple examples, and learn the basics of how R works with SQL Server. For your next step, see the following links:

- [Quickstart: Run R in T-SQL](#)
- [Tutorial: In-database analytics for R developers](#)

Python developers can learn how to use Python with SQL Server by following these tutorials:

- [Python tutorial: Predict ski rental with linear regression in SQL Server Machine Learning Services](#)
- [Python tutorial: Categorizing customers using k-means clustering with SQL Server Machine Learning Services](#)

Quickstart: Run simple Python scripts with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In this quickstart, you'll run a set of simple Python scripts using [SQL Server Machine Learning Services](#), [Azure SQL Managed Instance Machine Learning Services](#), or [SQL Server Big Data Clusters](#). You'll learn how to use the stored procedure `sp_execute_external_script` to execute the script in a SQL Server instance.

Prerequisites

You need the following prerequisites to run this quickstart.

- A SQL database on one of these platforms:
 - [SQL Server Machine Learning Services](#). To install, see the [Windows installation guide](#) or the [Linux installation guide](#).
 - SQL Server 2019 Big Data Clusters. See how to [enable Machine Learning Services on SQL Server 2019 Big Data Clusters](#).
 - Azure SQL Managed Instance Machine Learning Services. For information, see the [Azure SQL Managed Instance Machine Learning Services overview](#).
- A tool for running SQL queries that contain Python scripts. This quickstart uses [Azure Data Studio](#).

Run a simple script

To run a Python script, you'll pass it as an argument to the system stored procedure, `sp_execute_external_script`. This system stored procedure starts the Python runtime in the context of SQL machine learning, passes data to Python, manages Python user sessions securely, and returns any results to the client.

In the following steps, you'll run this example Python script in your database:

```
Python
```

```
a = 1
b = 2
c = a/b
```



```
d = a*b
print(c, d)
```

1. Open a new query window in **Azure Data Studio** connected to your SQL instance.
2. Pass the complete Python script to the `sp_execute_external_script` stored procedure.

The script is passed through the `@script` argument. Everything inside the `@script` argument must be valid Python code.

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'
      , @script = N'
a = 1
b = 2
c = a/b
d = a*b
print(c, d)
'
```

3. The correct result is calculated and the Python `print` function returns the result to the **Messages** window.

It should look something like this.

Results

text

```
STDOUT message(s) from external script:
0.5 2
```

Run a Hello World script

A typical example script is one that just outputs the string "Hello World". Run the following command.

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'
      , @script = N'OutputDataSet = InputDataSet'
      , @input_data_1 = N'SELECT 1 AS hello'
WITH RESULT SETS([[Hello World] INT]);
GO
```

Inputs to the `sp_execute_external_script` stored procedure include:

Input	Description
@language	defines the language extension to call, in this case Python
@script	defines the commands passed to the Python runtime. Your entire Python script must be enclosed in this argument, as Unicode text. You could also add the text to a variable of type <code>nvarchar</code> and then call the variable
@input_data_1	data returned by the query, passed to the Python runtime, which returns the data as a data frame
WITH RESULT SETS	clause defines the schema of the returned data table for SQL machine learning, adding "Hello World" as the column name, <code>int</code> for the data type

The command outputs the following text:

Hello World
1

Use inputs and outputs

By default, `sp_execute_external_script` accepts a single dataset as input, which typically you supply in the form of a valid SQL query. It then returns a single Python data frame as output.

For now, let's use the default input and output variables of `sp_execute_external_script`: `InputDataSet` and `OutputDataSet`.

1. Create a small table of test data.

SQL

```
CREATE TABLE PythonTestData (col1 INT NOT NULL)

INSERT INTO PythonTestData
VALUES (1);

INSERT INTO PythonTestData
VALUES (10);

INSERT INTO PythonTestData
VALUES (100);

GO
```

- Use the `SELECT` statement to query the table.

SQL

```
SELECT *  
FROM PythonTestData
```

Results

	col1
1	1
2	10
3	100

- Run the following Python script. It retrieves the data from the table using the `SELECT` statement, passes it through the Python runtime, and returns the data as a data frame. The `WITH RESULT SETS` clause defines the schema of the returned data table for SQL, adding the column name *NewColName*.

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'  
    , @script = N'OutputDataSet = InputDataSet;'  
    , @input_data_1 = N'SELECT * FROM PythonTestData;'  
WITH RESULT SETS(([NewColName] INT NOT NULL));
```

Results

	NewColName
1	1
2	10
3	100

- Now change the names of the input and output variables. The default input and output variable names are `InputDataSet` and `OutputDataSet`, the following script changes the names to `SQL_in` and `SQL_out`:

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'  
    , @script = N'SQL_out = SQL_in;'  
    , @input_data_1 = N'SELECT 12 as Col;'  
    , @input_data_1_name = N'SQL_in'  
    , @output_data_1_name = N'SQL_out'  
WITH RESULT SETS(([NewColName] INT NOT NULL));
```

Note that Python is case-sensitive. The input and output variables used in the Python script (**SQL_out**, **SQL_in**) need to match the names defined with `@input_data_1_name` and `@output_data_1_name`, including case.

💡 Tip

Only one input dataset can be passed as a parameter, and you can return only one dataset. However, you can call other datasets from inside your Python code and you can return outputs of other types in addition to the dataset. You can also add the **OUTPUT** keyword to any parameter to have it returned with the results.

5. You can also generate values just using the Python script with no input data (`@input_data_1` is set to blank).

The following script outputs the text "hello" and "world".

```
SQL

EXECUTE sp_execute_external_script @language = N'Python'
    , @script = N'
import pandas as pd
mytextvariable = pandas.Series(["hello", " ", "world"]);
OutputDataSet = pd.DataFrame(mytextvariable);
'
    , @input_data_1 = N''
WITH RESULT SETS(((Col1] CHAR(20) NOT NULL));
```

Results

	Col1
1	hello
2	
3	world

@script as input" />

💡 Tip

Python uses leading spaces to group statements. So when the imbedded Python script spans multiple lines, as in the preceding script, don't try to indent the Python commands to be in line with the SQL commands. For example, **this script will produce an error:**

```
SQL
```

```
EXECUTE sp_execute_external_script @language = N'Python'
, @script = N'
import pandas as pd
mytextvariable = pandas.Series(["hello", " ", "world"]);
OutputDataSet = pd.DataFrame(mytextvariable);
'
, @input_data_1 = N''
WITH RESULT SETS(([Col1] CHAR(20) NOT NULL));
```

Check Python version

If you would like to see which version of Python is installed in your server, run the following script.

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'
, @script = N'
import sys
print(sys.version)
'
GO
```

The Python `print` function returns the version to the **Messages** window. In the example output below, you can see that in this case, Python version 3.5.2 is installed.

Results

text

```
STDOUT message(s) from external script:
3.5.2 |Continuum Analytics, Inc.| (default, Jul 5 2016, 11:41:13) [MSC
v.1900 64 bit (AMD64)]
```

List Python packages

Microsoft provides a number of Python packages pre-installed with Machine Learning Services in SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x). In SQL Server 2022 (16.x), you can download and install any custom Python runtimes and packages as desired.

To see a list of which Python packages are installed, including version, run the following script.

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'  
    , @script = N'  
import pkg_resources  
import pandas  
dists = [str(d) for d in pkg_resources.working_set]  
OutputDataSet = pandas.DataFrame(dists)  
'  
WITH RESULT SETS(([Package] NVARCHAR(max)))  
GO
```

The list is from `pkg_resources.working_set` in Python and returned to SQL as a data frame.

Next steps

To learn how to use data structures when using Python in SQL machine learning, follow this quickstart:

[Quickstart: Data structures and objects using Python](#)

Quickstart: Data structures and objects using Python with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In this quickstart, you'll learn how to use data structures and data types when using Python in [SQL Server Machine Learning Services](#), [Azure SQL Managed Instance Machine Learning Services](#), or on [SQL Server Big Data Clusters](#). You'll learn about moving data between Python and SQL Server, and the common issues that might occur.

SQL machine learning relies on the Python **pandas** package, which is great for working with tabular data. However, you cannot pass a scalar from Python to your database and expect it to *just work*. In this quickstart, you'll review some basic data structure definitions, to prepare you for additional issues that you might run across when passing tabular data between Python and the database.

Concepts to know up front include:

- A data frame is a table with *multiple* columns.
- A single column of a data frame is a list-like object called a series.
- A single value of a data frame is called a cell and is accessed by index.

How would you expose the single result of a calculation as a data frame, if a data.frame requires a tabular structure? One answer is to represent the single scalar value as a series, which is easily converted to a data frame.

Note

When returning dates, Python in SQL uses DATETIME which has a restricted date range of 1753-01-01(-53690) through 9999-12-31(2958463).

Prerequisites

You need the following prerequisites to run this quickstart.

- A SQL database on one of these platforms:
 - [SQL Server Machine Learning Services](#). To install, see the [Windows installation guide](#) or the [Linux installation guide](#).

- SQL Server Big Data Clusters. See how to [enable Machine Learning Services on SQL Server Big Data Clusters](#).
- Azure SQL Managed Instance Machine Learning Services. For information, see the [Azure SQL Managed Instance Machine Learning Services overview](#).
- A tool for running SQL queries that contain Python scripts. This quickstart uses [Azure Data Studio](#).

Scalar value as a series

This example does some simple math and converts a scalar into a series.

1. A series requires an index, which you can assign manually, as shown here, or programmatically.

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'  
    , @script = N'  
a = 1  
b = 2  
c = a/b  
print(c)  
s = pandas.Series(c, index=["simple math example 1"])  
print(s)  
'
```

Because the series hasn't been converted to a data.frame, the values are returned in the Messages window, but you can see that the results are in a more tabular format.

Results

text

```
STDOUT message(s) from external script:  
0.5  
simple math example 1    0.5  
dtype: float64
```

2. To increase the length of the series, you can add new values, using an array.

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'  
    , @script = N'
```



```
a = 1
b = 2
c = a/b
d = a*b
s = pandas.Series([c,d])
print(s)
'
```

If you do not specify an index, an index is generated that has values starting with 0 and ending with the length of the array.

Results

text

```
STDOUT message(s) from external script:
0    0.5
1    2.0
dtype: float64
```

3. If you increase the number of **index** values, but don't add new **data** values, the data values are repeated to fill the series.

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'
    , @script = N'
a = 1
b = 2
c = a/b
s = pandas.Series(c, index=["simple math example 1", "simple math
example 2"])
print(s)
'
```

Results

text

```
STDOUT message(s) from external script:
0.5
simple math example 1    0.5
simple math example 2    0.5
dtype: float64
```

Convert series to data frame

Having converted the scalar math results to a tabular structure, you still need to convert them to a format that SQL machine learning can handle.

1. To convert a series to a data.frame, call the pandas [DataFrame](#) method.

```
SQL

EXECUTE sp_execute_external_script @language = N'Python'
    , @script = N'
import pandas as pd
a = 1
b = 2
c = a/b
d = a*b
s = pandas.Series([c,d])
print(s)
df = pd.DataFrame(s)
OutputDataSet = df
'

WITH RESULT SETS((ResultValue FLOAT))
```

The result is shown below. Even if you use the index to get specific values from the data.frame, the index values aren't part of the output.

Results

ResultValue
0.5
2

Output values into data.frame

Now you'll output specific values from two series of math results in a data.frame. The first has an index of sequential values generated by Python. The second uses an arbitrary index of string values.

1. The following example gets a value from the series using an integer index.

```
SQL

EXECUTE sp_execute_external_script @language = N'Python'
    , @script = N'
import pandas as pd
a = 1
b = 2
c = a/b
```

```

d = a*b
s = pandas.Series([c,d])
print(s)
df = pd.DataFrame(s, index=[1])
OutputDataSet = df
'
WITH RESULT SETS((ResultValue FLOAT))

```

Results

ResultValue
2.0

Remember that the auto-generated index starts at 0. Try using an out of range index value and see what happens.

- Now get a single value from the other data frame using a string index.

SQL

```

EXECUTE sp_execute_external_script @language = N'Python'
    , @script = N'
import pandas as pd
a = 1
b = 2
c = a/b
s = pandas.Series(c, index=["simple math example 1", "simple math
example 2"])
print(s)
df = pd.DataFrame(s, index=["simple math example 1"])
OutputDataSet = df
'
WITH RESULT SETS((ResultValue FLOAT))

```

Results

ResultValue
0.5

If you try to use a numeric index to get a value from this series, you get an error.

Next steps

To learn about writing advanced Python functions with SQL machine learning, follow this quickstart:

Write advanced Python functions

Quickstart: Python functions with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In this quickstart, you'll learn how to use Python mathematical and utility functions with [SQL Server Machine Learning Services](#), [Azure SQL Managed Instance Machine Learning Services](#), or [SQL Server Big Data Clusters](#). Statistical functions are often complicated to implement in T-SQL, but can be done in Python with only a few lines of code.

Prerequisites

You need the following prerequisites to run this quickstart.

- A SQL database on one of these platforms:
 - [SQL Server Machine Learning Services](#). To install, see the [Windows installation guide](#) or the [Linux installation guide](#).
 - SQL Server Big Data Clusters. See how to [enable Machine Learning Services on SQL Server Big Data Clusters](#).
 - Azure SQL Managed Instance Machine Learning Services. For information, see the [Azure SQL Managed Instance Machine Learning Services overview](#).
- A tool for running SQL queries that contain Python scripts. This quickstart uses [Azure Data Studio](#).

Create a stored procedure to generate random numbers

For simplicity, let's use the Python `numpy` package, that's installed and loaded by default. The package contains hundreds of functions for common statistical tasks, among them the `random.normal` function, which generates a specified number of random numbers using the normal distribution, given a standard deviation and mean.

For example, the following Python code returns 100 numbers on a mean of 50, given a standard deviation of 3.

```
Python
```

```
numpy.random.normal(size=100, loc=50, scale=3)
```

To call this line of Python from T-SQL, add the Python function in the Python script parameter of `sp_execute_external_script`. The output expects a data frame, so use `pandas` to convert it.

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'
    , @script = N'
import numpy
import pandas
OutputDataSet = pandas.DataFrame(numpy.random.normal(size=100, loc=50,
scale=3));
'
    , @input_data_1 = N'    ';
WITH RESULT SETS([[Density] FLOAT NOT NULL]);
```

What if you'd like to make it easier to generate a different set of random numbers? You define a stored procedure that gets the arguments from the user, then pass those arguments into the Python script as variables.

SQL

```
CREATE PROCEDURE MyPyNorm (
    @param1 INT
    , @param2 INT
    , @param3 INT
)
AS
EXECUTE sp_execute_external_script @language = N'Python'
    , @script = N'
import numpy
import pandas
OutputDataSet = pandas.DataFrame(numpy.random.normal(size=mynumbers,
loc=mymean, scale=mysd));
'
    , @input_data_1 = N'    ';
    , @params = N' @mynumbers int, @mymean int, @mysd int'
    , @mynumbers = @param1
    , @mymean = @param2
    , @mysd = @param3
WITH RESULT SETS([[Density] FLOAT NOT NULL]);
```

- The first line defines each of the SQL input parameters that are required when the stored procedure is executed.
- The line beginning with `@params` defines all variables used by the Python code, and the corresponding SQL data types.

- The lines that immediately follow map the SQL parameter names to the corresponding Python variable names.

Now that you've wrapped the Python function in a stored procedure, you can easily call the function and pass in different values, like this:

SQL

```
EXECUTE MyPyNorm @param1 = 100,@param2 = 50, @param3 = 3
```

Use Python utility functions for troubleshooting

Python packages provide a variety of utility functions for investigating the current Python environment. These functions can be useful if you're finding discrepancies in the way your Python code performs in SQL Server and in outside environments.

For example, you might use system timing functions in the `time` package to measure the amount of time used by Python processes and analyze performance issues.

SQL

```
EXECUTE sp_execute_external_script
    @language = N'Python'
    , @script = N'
import time
start_time = time.time()

# Run Python processes

elapsed_time = time.time() - start_time
'
    , @input_data_1 = N' ;';
```

Next steps

To create a machine learning model using Python with SQL machine learning, follow this quickstart:

[Quickstart: Create and score a predictive model in Python](#)

Quickstart: Create and score a predictive model in Python with SQL machine learning




Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In this quickstart, you'll create and train a predictive model using Python. You'll save the model to a table in your SQL Server instance, and then use the model to predict values from new data using [SQL Server Machine Learning Services](#), [Azure SQL Managed Instance Machine Learning Services](#), or [SQL Server Big Data Clusters](#).

You'll create and execute two stored procedures running in SQL. The first one uses the classic Iris flower data set and generates a Naïve Bayes model to predict an Iris species based on flower characteristics. The second procedure is for scoring - it calls the model generated in the first procedure to output a set of predictions based on new data. By placing Python code in a SQL stored procedure, operations are contained in SQL, are reusable, and can be called by other stored procedures and client applications.

By completing this quickstart, you'll learn:

-  How to embed Python code in a stored procedure
-  How to pass inputs to your code through inputs on the stored procedure
-  How stored procedures are used to operationalize models

Prerequisites

You need the following prerequisites to run this quickstart.

- A SQL database on one of these platforms:
 - [SQL Server Machine Learning Services](#). To install, see the [Windows installation guide](#) or the [Linux installation guide](#).
 - SQL Server Big Data Clusters. See how to [enable Machine Learning Services on SQL Server Big Data Clusters](#).
 - Azure SQL Managed Instance Machine Learning Services. For information, see the [Azure SQL Managed Instance Machine Learning Services overview](#).
- A tool for running SQL queries that contain Python scripts. This quickstart uses [Azure Data Studio](#).

- The sample data used in this exercise is the Iris sample data. Follow the instructions in [Iris demo data](#) to create the sample database `irissql`.

Create a stored procedure that generates models

In this step, you'll create a stored procedure that generates a model for predicting outcomes.

1. Open Azure Data Studio, connect to your SQL instance, and open a new query window.
2. Connect to the `irissql` database.

```
SQL

USE irissql
GO
```

3. Copy in the following code to create a new stored procedure.

When executed, this procedure calls `sp_execute_external_script` to start a Python session.

Inputs needed by your Python code are passed as input parameters on this stored procedure. Output will be a trained model, based on the Python `scikit-learn` library for the machine learning algorithm.

This code uses [pickle](#) to serialize the model. The model will be trained using data from columns 0 through 4 from the `iris_data` table.

The parameters you see in the second part of the procedure articulate data inputs and model outputs. As much as possible, you want the Python code running in a stored procedure to have clearly defined inputs and outputs that map to stored procedure inputs and outputs passed in at run time.

```
SQL

CREATE PROCEDURE generate_iris_model (@trained_model VARBINARY(max)
OUTPUT)
AS
BEGIN
    EXECUTE sp_execute_external_script @language = N'Python'
        , @script = N'
import pickle
```

```

from sklearn.naive_bayes import GaussianNB
GNB = GaussianNB()
trained_model = pickle.dumps(GNB.fit(iris_data[["Sepal.Length",
"Sepal.Width", "Petal.Length", "Petal.Width"]],
iris_data[["SpeciesId"]].values.ravel()))
'
    , @input_data_1 = N'select "Sepal.Length", "Sepal.Width",
"Petal.Length", "Petal.Width", "SpeciesId" from iris_data'
    , @input_data_1_name = N'iris_data'
    , @params = N'@trained_model varbinary(max) OUTPUT'
    , @trained_model = @trained_model OUTPUT;
END;
GO

```

4. Verify the stored procedure exists.

If the T-SQL script from the previous step ran without error, a new stored procedure called **generate_iris_model** is created and added to the **irissql** database. You can find stored procedures in the Azure Data Studio **Object Explorer**, under **Programmability**.

Execute the procedure to create and train models

In this step, you execute the procedure to run the embedded code, creating a trained and serialized model as an output.

Models that are stored for reuse in your database are serialized as a byte stream and stored in a VARBINARY(MAX) column in a database table. Once the model is created, trained, serialized, and saved to a database, it can be called by other procedures or by the **PREDICT T-SQL** function in scoring workloads.

1. Run the following script to execute the procedure. The specific statement for executing a stored procedure is **EXECUTE** on the fourth line.

This particular script deletes an existing model of the same name ("Naive Bayes") to make room for new ones created by rerunning the same procedure. Without model deletion, an error occurs stating the object already exists. The model is stored in a table called **iris_models**, provisioned when you created the **irissql** database.

SQL

```

DECLARE @model varbinary(max);
DECLARE @new_model_name varchar(50)

```

```

SET @new_model_name = 'Naive Bayes'
EXECUTE generate_iris_model @model OUTPUT;
DELETE iris_models WHERE model_name = @new_model_name;
INSERT INTO iris_models (model_name, model) values(@new_model_name,
@model);
GO

```

2. Verify that the model was inserted.

SQL

```
SELECT * FROM dbo.iris_models
```

Results

model_name	model
Naive Bayes	0x800363736B6C6561726E2E6E616976655F62617965730A...

Create and execute a stored procedure for generating predictions

Now that you have created, trained, and saved a model, move on to the next step: creating a stored procedure that generates predictions. You'll do this by calling `sp_execute_external_script` to run a Python script that loads the serialized model and gives it new data inputs to score.

1. Run the following code to create the stored procedure that performs scoring. At run time, this procedure will load a binary model, use columns `[1,2,3,4]` as inputs, and specify columns `[0,5,6]` as output.

SQL

```

CREATE PROCEDURE predict_species (@model VARCHAR(100))
AS
BEGIN
    DECLARE @nb_model VARBINARY(max) = (
        SELECT model
        FROM iris_models
        WHERE model_name = @model
    );

    EXECUTE sp_execute_external_script @language = N'Python'
        , @script = N'
import pickle
irismodel = pickle.loads(nb_model)

```

```

species_pred = irismodel.predict(iris_data[["Sepal.Length",
"Sepal.Width", "Petal.Length", "Petal.Width"]])
iris_data["PredictedSpecies"] = species_pred
OutputDataSet = iris_data[["id","SpeciesId","PredictedSpecies"]]
print(OutputDataSet)
'
    , @input_data_1 = N'select id, "Sepal.Length", "Sepal.Width",
"Petal.Length", "Petal.Width", "SpeciesId" from iris_data'
    , @input_data_1_name = N'iris_data'
    , @params = N'@nb_model varbinary(max)'
    , @nb_model = @nb_model
    WITH RESULT SETS((
        "id" INT
        , "SpeciesId" INT
        , "SpeciesId.Predicted" INT
    ));
END;
GO

```

- Execute the stored procedure, giving the model name "Naive Bayes" so that the procedure knows which model to use.

```

SQL

EXECUTE predict_species 'Naive Bayes';
GO

```

When you run the stored procedure, it returns a Python data.frame. This line of T-SQL specifies the schema for the returned results: `WITH RESULT SETS (("id" int, "SpeciesId" int, "SpeciesId.Predicted" int));`. You can insert the results into a new table, or return them to an application.

	id	SpeciesId	SpeciesId.Predicted
99	99	1	1
100	100	1	1
101	101	2	2
102	102	2	2
103	103	2	2
104	104	2	2
105	105	2	2
106	106	2	2
107	107	2	1
108	108	2	2
109	109	2	2

Query executed successfully. | ST-MSFT4 (14.0 RTM) | sqlpy | 00:00:22 | 150 rows

The results are 150 predictions about species using floral characteristics as inputs. For the majority of the observations, the predicted species matches the actual species.

This example has been made simple by using the Python iris dataset for both training and scoring. A more typical approach would involve running a SQL query to get the new data, and passing that into Python as `InputDataSet`.

Conclusion

In this exercise, you learned how to create stored procedures dedicated to different tasks, where each stored procedure used the system stored procedure `sp_execute_external_script` to start a Python process. Inputs to the Python process are passed to `sp_execute_external` as parameters. Both the Python script itself and data variables in a database are passed as inputs.

Generally, you should only plan on using Azure Data Studio with polished Python code, or simple Python code that returns row-based output. As a tool, Azure Data Studio supports query languages like T-SQL and returns flattened rowsets. If your code generates visual output like a scatterplot or histogram, you need a separate tool or end-user application that can render the image outside of the stored procedure.

For some Python developers who are used to writing all-inclusive script handling a range of operations, organizing tasks into separate procedures might seem unnecessary. But training and scoring have different use cases. By separating them, you can put each task on a different schedule and scope permissions to each operation.

A final benefit is that the processes can be modified using parameters. In this exercise, Python code that created the model (named "Naive Bayes" in this example) was passed as an input to a second stored procedure calling the model in a scoring process. This exercise only uses one model, but you can imagine how parameterizing the model in a scoring task would make that script more useful.

Next steps

For more information on tutorials for Python with SQL machine learning, see:

- [Python tutorials](#)

Quickstart: Run simple R scripts with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In this quickstart, you'll run a set of simple R scripts using [SQL Server Machine Learning Services](#). You'll learn how to use the stored procedure `sp_execute_external_script` to execute the script in a SQL Server instance.

Prerequisites

You need the following prerequisites to run this quickstart.

- SQL Server Machine Learning Services. To install Machine Learning Services, see the [Windows installation guide](#).
- A tool for running SQL queries that contain R scripts. This quickstart uses [Azure Data Studio](#).

Run a simple script

To run an R script, you'll pass it as an argument to the system stored procedure, `sp_execute_external_script`. This system stored procedure starts the R runtime, passes data to R, manages R user sessions securely, and returns any results to the client.

In the following steps, you'll run this example R script:

```
R

a <- 1
b <- 2
c <- a/b
d <- a*b
print(c(c, d))
```

1. Open **Azure Data Studio** and connect to your server.
2. Pass the complete R script to the `sp_execute_external_script` stored procedure.

The script is passed through the `@script` argument. Everything inside the `@script` argument must be valid R code.

SQL

```
EXECUTE sp_execute_external_script @language = N'R'
    , @script = N'
a <- 1
b <- 2
c <- a/b
d <- a*b
print(c(c, d))
'
```

3. The correct result is calculated and the R `print` function returns the result to the **Messages** window.

It should look something like this.

Results

text

```
STDOUT message(s) from external script:
0.5 2
```

Run a Hello World script

A typical example script is one that just outputs the string "Hello World". Run the following command.

SQL

```
EXECUTE sp_execute_external_script @language = N'R'
    , @script = N'OutputDataSet<-InputDataSet'
    , @input_data_1 = N'SELECT 1 AS hello'
WITH RESULT SETS([[Hello World] INT]);
GO
```

Inputs to the `sp_execute_external_script` stored procedure include:

Input	Description
@language	defines the language extension to call, in this case, R
@script	defines the commands passed to the R runtime. Your entire R script must be enclosed in this argument, as Unicode text. You could also add the text to a variable of type <code>nvarchar</code> and then call the variable

Input	Description
@input_data_1	data returned by the query, passed to the R runtime, which returns the data as a data frame
WITH RESULT SETS	clause defines the schema of the returned data table, adding "Hello World" as the column name, <code>int</code> for the data type

The command outputs the following text:

Hello World
1

Use inputs and outputs

By default, `sp_execute_external_script` accepts a single dataset as input, which typically you supply in the form of a valid SQL query. It then returns a single R data frame as output.

For now, let's use the default input and output variables of `sp_execute_external_script`: `InputDataSet` and `OutputDataSet`.

1. Create a small table of test data.

```
SQL

CREATE TABLE RTestData (col1 INT NOT NULL)

INSERT INTO RTestData
VALUES (1);

INSERT INTO RTestData
VALUES (10);

INSERT INTO RTestData
VALUES (100);
GO
```

2. Use the `SELECT` statement to query the table.

```
SQL

SELECT *
FROM RTestData
```


Results

	col1
1	1
2	10
3	100

- Run the following R script. It retrieves the data from the table using the `SELECT` statement, passes it through the R runtime, and returns the data as a data frame. The `WITH RESULT SETS` clause defines the schema of the returned data table for SQL, adding the column name *NewColName*.

SQL

```
EXECUTE sp_execute_external_script @language = N'R'  
    , @script = N'OutputDataSet <- InputDataSet;'  
    , @input_data_1 = N'SELECT * FROM RTestData;'  
WITH RESULT SETS(([NewColName] INT NOT NULL));
```

Results

	NewColName
1	1
2	10
3	100

- Now let's change the names of the input and output variables. The default input and output variable names are `InputDataSet` and `OutputDataSet`, this script changes the names to `SQL_in` and `SQL_out`:

SQL

```
EXECUTE sp_execute_external_script @language = N'R'  
    , @script = N' SQL_out <- SQL_in; '  
    , @input_data_1 = N' SELECT 12 as Col; '  
    , @input_data_1_name = N'SQL_in '  
    , @output_data_1_name = N'SQL_out '  
WITH RESULT SETS(([NewColName] INT NOT NULL));
```

Note that R is case-sensitive. The input and output variables used in the R script (`SQL_out`, `SQL_in`) need to match the names defined with `@input_data_1_name` and `@output_data_1_name`, including case.

 **Tip**

Only one input dataset can be passed as a parameter, and you can return only one dataset. However, you can call other datasets from inside your R code and you can return outputs of other types in addition to the dataset. You can also add the OUTPUT keyword to any parameter to have it returned with the results.

5. You also can generate values just using the R script with no input data (@input_data_1 is set to blank).

The following script outputs the text "hello" and "world".

```
SQL

EXECUTE sp_execute_external_script @language = N'R'
    , @script = N'
mytextvariable <- c("hello", " ", "world");
OutputDataSet <- as.data.frame(mytextvariable);
'
    , @input_data_1 = N''
WITH RESULT SETS(([Col1] CHAR(20) NOT NULL));
```

Results

	Col1
1	hello
2	
3	world

@script as input" />

Check R version

If you would like to see which version of R is installed, run the following script.

```
SQL

EXECUTE sp_execute_external_script @language = N'R'
    , @script = N'print(version)';
GO
```

The R `print` function returns the version to the **Messages** window. In the example output below, you can see that in this case, R version 3.4.4 is installed.

Results

text

STDOUT message(s) from external script:

```
platform      x86_64-w64-mingw32
arch          x86_64
os            mingw32
system        x86_64, mingw32
status
major         3
minor         4.4
year          2018
month         03
day           15
svn rev       74408
language      R
version.string R version 3.4.4 (2018-03-15)
nickname      Someone to Lean On
```

List R packages

Microsoft provides a number of R packages pre-installed with Machine Learning Services.

To see a list of which R packages are installed, including version, dependencies, license, and library path information, run the following script.

SQL

```
EXEC sp_execute_external_script @language = N'R'
    , @script = N'
OutputDataSet <- data.frame(installed.packages()[,c("Package", "Version",
"Depends", "License", "LibPath")]);'
WITH result sets((
    Package NVARCHAR(255)
    , Version NVARCHAR(100)
    , Depends NVARCHAR(4000)
    , License NVARCHAR(1000)
    , LibPath NVARCHAR(2000)
));
```

The output is from `installed.packages()` in R and is returned as a result set.

Results

	Package	Version	Depends	License	LibPath
1	base	3.3.3	NULL	Part of R 3.3.3	D:/SQL-SQL2017-R_SERVICES/library
2	boot	1.3-18	R (>= 3.0.0), graphics, stats	Unlimited	D:/SQL-SQL2017-R_SERVICES/library
3	checkpoint	0.4.0	R(>= 3.0.0)	GPL-2	D:/SQL-SQL2017-R_SERVICES/library
4	class	7.3-14	R (>= 3.0.0), stats, utils	GPL-2 GPL-3	D:/SQL-SQL2017-R_SERVICES/library
5	cluster	2.0.5	R (>= 3.0.1)	GPL (>= 2)	D:/SQL-SQL2017-R_SERVICES/library
6	codetools	0.2-15	R (>= 2.1)	GPL	D:/SQL-SQL2017-R_SERVICES/library
7	CompatibilityAPI	1.1.0	R (>= 3.2.2)	file LICENSE	D:/SQL-SQL2017-R_SERVICES/library
8	compiler	3.3.3	NULL	Part of R 3.3.3	D:/SQL-SQL2017-R_SERVICES/library
9	curl	2.6	R (>= 3.0.0)	MIT + file LICENSE	D:/SQL-SQL2017-R_SERVICES/library
10	datasets	3.3.3	NULL	Part of R 3.3.3	D:/SQL-SQL2017-R_SERVICES/library
11	deployRserve	9.0.0	R (>= 1.5.0)	GPL version 2	D:/SQL-SQL2017-R_SERVICES/library
12	doParallel	1.0.10	R (>= 2.14.0), foreach(>= 1.2.0), iterators(>= 1...	GPL-2	D:/SQL-SQL2017-R_SERVICES/library
13	doRSR	10.0.0	R (>= 2.5.0), foreach(>= 1.2.0), iterators(>= 1....	file LICENSE	D:/SQL-SQL2017-R_SERVICES/library
14	foreach	1.4.5	R (>= 2.5.0)	Apache License (== 2.0)	D:/SQL-SQL2017-R_SERVICES/library
15	foreign	0.8-67	R (>= 3.0.0)	GPL (>= 2)	D:/SQL-SQL2017-R_SERVICES/library
16	graphics	3.3.3	NULL	Part of R 3.3.3	D:/SQL-SQL2017-R_SERVICES/library



Next steps

To learn how to use data structures when using R with SQL machine learning, follow this quickstart:

[Handle data types and objects using R with SQL machine learning](#)

Quickstart: Data structures, data types, and objects using R with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In this quickstart, you'll learn how to use data structures and data types when using R in [SQL Server Machine Learning Services](#). You'll learn about moving data between R and SQL Server, and the common issues that might occur.

Common issues to know up front include:

- Data types sometimes don't match
- Implicit conversions might take place
- Cast and convert operations are sometimes required
- R and SQL use different data objects

Prerequisites

You need the following prerequisites to run this quickstart.

- SQL Server Machine Learning Services. To install Machine Learning Services, see the [Windows installation guide](#).
- A tool for running SQL queries that contain R scripts. This quickstart uses [Azure Data Studio](#).

Always return a data frame

When your script returns results from R to SQL Server, it must return the data as a **data.frame**. Any other type of object that you generate in your script - whether that be a list, factor, vector, or binary data - must be converted to a data frame if you want to output it as part of the stored procedure results. Fortunately, there are multiple R functions to support changing other objects to a data frame. You can even serialize a binary model and return it in a data frame, which you'll do later in this quickstart.

First, let's experiment with some R basic R objects - vectors, matrices, and lists - and see how conversion to a data frame changes the output passed to SQL Server.

Compare these two "Hello World" scripts in R. The scripts look almost identical, but the first returns a single column of three values, whereas the second returns three columns with a single value each.

Example 1

SQL

```
EXECUTE sp_execute_external_script
  @language = N'R'
  , @script = N' mytextvariable <- c("hello", " ", "world");
  OutputDataSet <- as.data.frame(mytextvariable);'
  , @input_data_1 = N' ';
```

Example 2

SQL

```
EXECUTE sp_execute_external_script
  @language = N'R'
  , @script = N' OutputDataSet<- data.frame(c("hello"), " ",
  c("world"));'
  , @input_data_1 = N' ';
```

Identify schema and data types

Why are the results so different?

The answer can usually be found by using the R `str()` command. Add the function `str(object_name)` anywhere in your R script to have the data schema of the specified R object returned as an informational message.

To figure out why Example 1 and Example 2 have such different results, insert the line `str(OutputDataSet)` at the end of the `@script` variable definition in each statement, like this:

Example 1 with str function added

SQL

```
EXECUTE sp_execute_external_script
  @language = N'R'
  , @script = N' mytextvariable <- c("hello", " ", "world");
  OutputDataSet <- as.data.frame(mytextvariable);
  str(OutputDataSet);'
```

```
, @input_data_1 = N' '  
;
```

Example 2 with str function added

SQL

```
EXECUTE sp_execute_external_script  
  @language = N'R',  
  @script = N' OutputDataSet <- data.frame(c("hello"), " ", c("world"));  
    str(OutputDataSet);' ,  
  @input_data_1 = N' ';
```

Now, review the text in **Messages** to see why the output is different.

Results - Example 1

SQL

```
STDOUT message(s) from external script:  
'data.frame':  3 obs. of  1 variable:  
 $ mytextvariable: Factor w/ 3 levels " ","hello","world": 2 1 3
```

Results - Example 2

SQL

```
STDOUT message(s) from external script:  
'data.frame':  1 obs. of  3 variables:  
 $ c..hello..: Factor w/ 1 level "hello": 1  
 $ X...      : Factor w/ 1 level " ": 1  
 $ c..world..: Factor w/ 1 level "world": 1
```

As you can see, a slight change in R syntax had a big effect on the schema of the results. We won't go into why, but the differences in R data types are explained in details in the *Data Structures* section in ["Advanced R" by Hadley Wickham](#).

For now, just be aware that you need to check the expected results when coercing R objects into data frames.

Tip

You can also use R identity functions, such as `is.matrix`, `is.vector`, to return information about the internal data structure.

Implicit conversion of data objects

Each R data object has its own rules for how values are handled when combined with other data objects if the two data objects have the same number of dimensions, or if any data object contains heterogeneous data types.

First, create a small table of test data.

SQL

```
CREATE TABLE RTestData (col1 INT NOT NULL)

INSERT INTO RTestData
VALUES (1);

INSERT INTO RTestData
VALUES (10);

INSERT INTO RTestData
VALUES (100);
GO
```

For example, assume you run the following statement to perform matrix multiplication using R. You multiply a single-column matrix with the three values by an array with four values, and expect a 4x3 matrix as a result.

SQL

```
EXECUTE sp_execute_external_script
  @language = N'R'
  , @script = N'
      x <- as.matrix(InputDataSet);
      y <- array(12:15);
      OutputDataSet <- as.data.frame(x %% y);'
  , @input_data_1 = N' SELECT [Col1] from RTestData;'
  WITH RESULT SETS (([Col1] int, [Col2] int, [Col3] int, Col4 int));
```

Under the covers, the column of three values is converted to a single-column matrix. Because a matrix is just a special case of an array in R, the array `y` is implicitly coerced to a single-column matrix to make the two arguments conform.

Results

Col1	Col2	Col3	Col4
12	13	14	15

Col1	Col2	Col3	Col4
120	130	140	150
1200	1300	1400	1500

However, note what happens when you change the size of the array `y`.

SQL

```
execute sp_execute_external_script
  @language = N'R'
  , @script = N'
      x <- as.matrix(InputDataSet);
      y <- array(12:14);
      OutputDataSet <- as.data.frame(y %*% x);'
  , @input_data_1 = N' SELECT [Col1] from RTestData;'
  WITH RESULT SETS ([[Col1] int ]);
```

Now R returns a single value as the result.

Results

Col1
1542

Why? In this case, because the two arguments can be handled as vectors of the same length, R returns the inner product as a matrix. This is the expected behavior according to the rules of linear algebra; however, it could cause problems if your downstream application expects the output schema to never change!

Tip

Getting errors? Make sure that you're running the stored procedure in the context of the database that contains the table, and not in **master** or another database.

Also, we suggest that you avoid using temporary tables for these examples. Some R clients will terminate a connection between batches, deleting temporary tables.

Merge or multiply columns of different length

R provides great flexibility for working with vectors of different sizes, and for combining these column-like structures into data frames. Lists of vectors can look like a table, but

they don't follow all the rules that govern database tables.

For example, the following script defines a numeric array of length 6 and stores it in the R variable `df1`. The numeric array is then combined with the integers of the `RTestData` table, which contains three (3) values, to make a new data frame, `df2`.

SQL

```
EXECUTE sp_execute_external_script
  @language = N'R'
  , @script = N'
      df1 <- as.data.frame( array(1:6) );
      df2 <- as.data.frame( c( InputDataSet , df1 ));
      OutputDataSet <- df2'
  , @input_data_1 = N' SELECT [Col1] from RTestData;'
  WITH RESULT SETS (( [Col2] int not null, [Col3] int not null ));
```

To fill out the data frame, R repeats the elements retrieved from `RTestData` as many times as needed to match the number of elements in the array `df1`.

Results

Col2	Col3
1	1
10	2
100	3
1	4
10	5
100	6

Remember that a data frame only looks like a table, and is actually a list of vectors.

Cast or convert data

R and SQL Server don't use the same data types, so when you run a query in SQL Server to get data and then pass that to the R runtime, some type of implicit conversion usually takes place. Another set of conversions takes place when you return data from R to SQL Server.

- SQL Server pushes the data from the query to the R process managed by the Launchpad service and converts it to an internal representation for greater

efficiency.

- The R runtime loads the data into a `data.frame` variable and performs its own operations on the data.
- The database engine returns the data to SQL Server using a secured internal connection and presents the data in terms of SQL Server data types.
- You get the data by connecting to SQL Server using a client or network library that can issue SQL queries and handle tabular data sets. This client application can potentially affect the data in other ways.

To see how this works, run a query such as this one on the [AdventureWorksDW](#) data warehouse. This view returns sales data used in creating forecasts.

SQL

```
USE AdventureWorksDW
GO

SELECT ReportingDate
       , CAST(ModelRegion as varchar(50)) as ProductSeries
       , Amount
  FROM [AdventureWorksDW].[dbo].[vTimeSeries]
 WHERE [ModelRegion] = 'M200 Europe'
 ORDER BY ReportingDate ASC
```

ⓘ Note

You can use any version of AdventureWorks, or create a different query using a database of your own. The point is to try to handle some data that contains text, datetime and numeric values.

Now, try pasting this query as the input to the stored procedure.

SQL

```
EXECUTE sp_execute_external_script
  @language = N'R'
  , @script = N' str(InputDataSet);
  OutputDataSet <- InputDataSet;'
  , @input_data_1 = N'
    SELECT ReportingDate
       , CAST(ModelRegion as varchar(50)) as ProductSeries
       , Amount
  FROM [AdventureWorksDW].[dbo].[vTimeSeries]
  WHERE [ModelRegion] = ''M200 Europe''
  ORDER BY ReportingDate ASC ;'
WITH RESULT SETS undefined;
```

If you get an error, you'll probably need to make some edits to the query text. For example, the string predicate in the `WHERE` clause must be enclosed by two sets of single quotation marks.

After you get the query working, review the results of the `str` function to see how R treats the input data.

Results

```
text
STDOUT message(s) from external script: 'data.frame':  37 obs. of  3
variables:
STDOUT message(s) from external script: $ ReportingDate: POSIXct, format:
"2010-12-24 23:00:00" "2010-12-24 23:00:00"
STDOUT message(s) from external script: $ ProductSeries: Factor w/ 1 levels
"M200 Europe",..: 1 1 1 1 1 1 1 1 1 1
STDOUT message(s) from external script: $ Amount      : num  3400 16925
20350 16950 16950
```

- The datetime column has been processed using the R data type, `POSIXct`.
- The text column "ProductSeries" has been identified as a **factor**, meaning a categorical variable. String values are handled as factors by default. If you pass a string to R, it is converted to an integer for internal use, and then mapped back to the string on output.

Summary

From even these short examples, you can see the need to check the effects of data conversion when passing SQL queries as input. Because some SQL Server data types are not supported by R, consider these ways to avoid errors:

- Test your data in advance and verify columns or values in your schema that could be a problem when passed to R code.
- Specify columns in your input data source individually, rather than using `SELECT *`, and know how each column will be handled.
- Perform explicit casts as necessary when preparing your input data, to avoid surprises.
- Avoid passing columns of data (such as GUIDs or rowguids) that cause errors and aren't useful for modeling.

For more information on supported and unsupported data types, see [R libraries and data types](#).

Next steps

To learn about writing advanced R functions with SQL machine learning, follow this quickstart:

[Write advanced R functions with SQL machine learning](#)

Quickstart: R functions with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In this quickstart, you'll learn how to use R mathematical and utility functions with [SQL Server Machine Learning Services](#). Statistical functions are often complicated to implement in T-SQL, but can be done in R with only a few lines of code.

Prerequisites

You need the following prerequisites to run this quickstart.

- SQL Server Machine Learning Services. To install Machine Learning Services, see the [Windows installation guide](#).
- A tool for running SQL queries that contain R scripts. This quickstart uses [Azure Data Studio](#).

Create a stored procedure to generate random numbers

For simplicity, let's use the R `stats` package, that's installed and loaded by default. The package contains hundreds of functions for common statistical tasks, among them the `rnorm` function, which generates a specified number of random numbers using the normal distribution, given a standard deviation and mean.

For example, the following R code returns 100 numbers on a mean of 50, given a standard deviation of 3.

R

```
as.data.frame(rnorm(100, mean = 50, sd = 3));
```

To call this line of R from T-SQL, add the R function in the R script parameter of `sp_execute_external_script`, like this:

SQL

```
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N'
        OutputDataSet <- as.data.frame(rnorm(100, mean = 50, sd =3));'
    , @input_data_1 = N'    ';
    WITH RESULT SETS ([[Density] float NOT NULL]);
```

What if you'd like to make it easier to generate a different set of random numbers?

That's easy when combined with T-SQL. You define a stored procedure that gets the arguments from the user, then pass those arguments into the R script as variables.

SQL

```
CREATE PROCEDURE MyRNorm (
    @param1 INT
    , @param2 INT
    , @param3 INT
)
AS
EXECUTE sp_execute_external_script @language = N'R'
    , @script = N'
        OutputDataSet <- as.data.frame(rnorm(mynumbers, mymean, mysd));'
    , @input_data_1 = N'    ';
    , @params = N' @mynumbers int, @mymean int, @mysd int'
    , @mynumbers = @param1
    , @mymean = @param2
    , @mysd = @param3
WITH RESULT SETS([[Density] FLOAT NOT NULL]);
```

- The first line defines each of the SQL input parameters that are required when the stored procedure is executed.
- The line beginning with `@params` defines all variables used by the R code, and the corresponding SQL data types.
- The lines that immediately follow map the SQL parameter names to the corresponding R variable names.

Now that you've wrapped the R function in a stored procedure, you can easily call the function and pass in different values, like this:

SQL

```
EXECUTE MyRNorm @param1 = 100,@param2 = 50, @param3 = 3
```

Use R utility functions for troubleshooting

The `utils` package, installed by default, provides a variety of utility functions for investigating the current R environment. These functions can be useful if you're finding discrepancies in the way your R code performs in SQL Server and in outside environments.

For example, you might use the system timing functions in R, such as `system.time` and `proc.time`, to capture the time used by R processes and analyze performance issues. For an example, see the tutorial [Create Data Features](#) where R timing functions are embedded in the solution.

SQL

```
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N'
        library(utils);
        start.time <- proc.time();

        # Run R processes

        elapsed_time <- proc.time() - start.time;'
```

For other useful functions, see [Use R code profiling functions to improve performance](#).

Next steps

To create a machine learning model using R with SQL machine learning, follow this quickstart:

[Create and score a predictive model in R with SQL machine learning](#)

Quickstart: Create and score a predictive model in R with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)




In this quickstart, you'll create and train a predictive model using T. You'll save the model to a table in your SQL Server instance, and then use the model to predict values from new data using [SQL Server Machine Learning Services](#).

You'll create and execute two stored procedures running in SQL. The first one uses the `mtcars` dataset included with R and generates a simple generalized linear model (GLM) that predicts the probability that a vehicle has been fitted with a manual transmission. The second procedure is for scoring - it calls the model generated in the first procedure to output a set of predictions based on new data. By placing R code in a SQL stored procedure, operations are contained in SQL, are reusable, and can be called by other stored procedures and client applications.

Tip

If you need a refresher on linear models, try this tutorial which describes the process of fitting a model using `rxLinMod`: [Fitting Linear Models](#)

By completing this quickstart, you'll learn:

-  How to embed R code in a stored procedure
-  How to pass inputs to your code through inputs on the stored procedure
-  How stored procedures are used to operationalize models

Prerequisites

You need the following prerequisites to run this quickstart.

- SQL Server Machine Learning Services. To install Machine Learning Services, see the [Windows installation guide](#).
- A tool for running SQL queries that contain R scripts. This quickstart uses [Azure Data Studio](#).

Create the model

To create the model, you'll create source data for training, create the model and train it using the data, then store the model in a database where it can be used to generate predictions with new data.

Create the source data

1. Open Azure Data Studio, connect to your instance, and open a new query window.
2. Create a table to save the training data.

SQL

```
CREATE TABLE dbo.MTCars(  
  mpg decimal(10, 1) NOT NULL,  
  cyl int NOT NULL,  
  disp decimal(10, 1) NOT NULL,  
  hp int NOT NULL,  
  drat decimal(10, 2) NOT NULL,  
  wt decimal(10, 3) NOT NULL,  
  qsec decimal(10, 2) NOT NULL,  
  vs int NOT NULL,  
  am int NOT NULL,  
  gear int NOT NULL,  
  carb int NOT NULL  
);
```

3. Insert the data from the built-in dataset `mtcars`.

SQL

```
INSERT INTO dbo.MTCars  
EXEC sp_execute_external_script @language = N'R'  
  , @script = N'MTCars <- mtcars;'  
  , @input_data_1 = N''  
  , @output_data_1_name = N'MTCars';
```

Tip

Many datasets, small and large, are included with the R runtime. To get a list of datasets installed with R, type `library(help="datasets")` from an R command prompt.

Create and train the model

The car speed data contains two columns, both numeric: horsepower (`hp`) and weight (`wt`). From this data, you'll create a generalized linear model (GLM) that estimates the probability that a vehicle has been fitted with a manual transmission.

To build the model, you define the formula inside your R code, and pass the data as an input parameter.

SQL

```
DROP PROCEDURE IF EXISTS generate_GLM;
GO
CREATE PROCEDURE generate_GLM
AS
BEGIN
    EXEC sp_execute_external_script
        @language = N'R'
        , @script = N'carsModel <- glm(formula = am ~ hp + wt, data =
MTCarsData, family = binomial);
            trained_model <- data.frame(payload = as.raw(serialize(carsModel,
connection=NULL)));';
        , @input_data_1 = N'SELECT hp, wt, am FROM MTCars'
        , @input_data_1_name = N'MTCarsData'
        , @output_data_1_name = N'trained_model'
        WITH RESULT SETS ((model VARBINARY(max)));
END;
GO
```

- The first argument to `glm` is the *formula* parameter, which defines `am` as dependent on `hp + wt`.
- The input data is stored in the variable `MTCarsData`, which is populated by the SQL query. If you don't assign a specific name to your input data, the default variable name would be *InputDataSet*.

Store the model in the database

Next, store the model in a database so you can use it for prediction or retrain it.

1. Create a table to store the model.

The output of an R package that creates a model is usually a binary object. Therefore, the table where you store the model must provide a column of `varbinary(max)` type.

SQL

```
CREATE TABLE GLM_models (  
    model_name varchar(30) not null default('default model') primary  
key,  
    model varbinary(max) not null  
);
```

2. Run the following Transact-SQL statement to call the stored procedure, generate the model, and save it to the table you created.

SQL

```
INSERT INTO GLM_models(model)  
EXEC generate_GLM;
```

Tip

If you run this code a second time, you get this error: "Violation of PRIMARY KEY constraint...Cannot insert duplicate key in object dbo.stopping_distance_models". One option for avoiding this error is to update the name for each new model. For example, you could change the name to something more descriptive, and include the model type, the day you created it, and so forth.

SQL

```
UPDATE GLM_models  
SET model_name = 'GLM_' + format(getdate(), 'yyyy.MM.HH.mm', 'en-gb')  
WHERE model_name = 'default model'
```

Score new data using the trained model

Scoring is a term used in data science to mean generating predictions, probabilities, or other values based on new data fed into a trained model. You'll use the model you created in the previous section to score predictions against new data.

Create a table of new data

First, create a table with new data.

SQL

```

CREATE TABLE dbo.NewMTCars(
  hp INT NOT NULL
  , wt DECIMAL(10,3) NOT NULL
  , am INT NULL
)
GO

INSERT INTO dbo.NewMTCars(hp, wt)
VALUES (110, 2.634)

INSERT INTO dbo.NewMTCars(hp, wt)
VALUES (72, 3.435)

INSERT INTO dbo.NewMTCars(hp, wt)
VALUES (220, 5.220)

INSERT INTO dbo.NewMTCars(hp, wt)
VALUES (120, 2.800)
GO

```

Predict manual transmission

To get predictions based on your model, write a SQL script that does the following:

1. Gets the model you want
2. Gets the new input data
3. Calls an R prediction function that is compatible with that model

Over time, the table might contain multiple R models, all built using different parameters or algorithms, or trained on different subsets of data. In this example, we'll use the model named `default model`.

SQL

```

DECLARE @glmmodel varbinary(max) =
  (SELECT model FROM dbo.GLM_models WHERE model_name = 'default model');

EXEC sp_execute_external_script
  @language = N'R'
  , @script = N'
      current_model <- unserialize(as.raw(glmmodel));
      new <- data.frame(NewMTCars);
      predicted.am <- predict(current_model, new, type = "response");
      str(predicted.am);
      OutputDataSet <- cbind(new, predicted.am);
      '
  , @input_data_1 = N'SELECT hp, wt FROM dbo.NewMTCars'
  , @input_data_1_name = N'NewMTCars'
  , @params = N'@glmmodel varbinary(max)'

```

```
, @glmmodel = @glmmodel  
WITH RESULT SETS ((new_hp INT, new_wt DECIMAL(10,3), predicted_am  
DECIMAL(10,3)));
```

The script above performs the following steps:

- Use a SELECT statement to get a single model from the table, and pass it as an input parameter.
- After retrieving the model from the table, call the `unserialize` function on the model.
- Apply the `predict` function with appropriate arguments to the model, and provide the new input data.

ⓘ Note

In the example, the `str` function is added during the testing phase, to check the schema of data being returned from R. You can remove the statement later.

The column names used in the R script are not necessarily passed to the stored procedure output. Here the WITH RESULTS clause is used to define some new column names.

Results

	new_hp	new_wt	predicted_am
1	110	2.634	0.827
2	72	3.435	0.002
3	220	5.220	0.000
4	120	2.800	0.642

It's also possible to use the [PREDICT \(Transact-SQL\)](#) statement to generate a predicted value or score based on a stored model.

Next steps

For more information on tutorials for R with SQL machine learning, see:

- [R tutorials](#)

Python tutorials for SQL machine learning

Article • 02/28/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

This article describes the Python tutorials and quickstarts for [SQL Server Machine Learning Services](#).

Python tutorials

Tutorial	Description
Predict ski rental with linear regression	Use Python and linear regression to predict the number of ski rentals. Use notebooks in Azure Data Studio for preparing data and training the model, and T-SQL for model deployment.
Categorizing customers using k-means clustering	Use Python to develop and deploy a K-Means clustering model to categorize customers. Use notebooks in Azure Data Studio for preparing data and training the model, and T-SQL for model deployment.
Create a model using revoscalepy	Demonstrates how to run code from a remote Python client using SQL Server as compute context. The tutorial creates a model using <code>rxLinMod</code> from the <code>revoscalepy</code> library.
Python data analytics for SQL developers	This end-to-end walkthrough demonstrates the process of building a complete Python solution using T-SQL.

Python quickstarts

If you are new to SQL machine learning, you can also try the Python quickstarts.

Quickstart	Description
Run simple Python scripts	Learn the basics of how to call Python in T-SQL using <code>sp_execute_external_script</code> .
Data structures and objects using Python	Shows how SQL uses the Python pandas package to handle data structures.
Create and score a predictive model in Python	Explains how to create, train, and use a Python model to make predictions from new data.

Next steps

- [Python extension in SQL Server](#)

Python tutorial: Predict ski rental with linear regression with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In this four-part tutorial series, you will use Python and linear regression in [SQL Server Machine Learning Services](#) to predict the number of ski rentals. The tutorial uses a [Python notebook in Azure Data Studio](#).

Imagine you own a ski rental business and you want to predict the number of rentals that you'll have on a future date. This information will help you get your stock, staff, and facilities ready.

In the first part of this series, you'll get set up with the prerequisites. In parts two and three, you'll develop some Python scripts in a notebook to prepare your data and train a machine learning model. Then, in part three, you'll run those Python scripts inside the database using T-SQL stored procedures.

In this article, you'll learn how to:

-  Import a sample database

In [part two](#), you'll learn how to load the data from a database into a Python data frame, and prepare the data in Python.

In [part three](#), you'll learn how to train a linear regression model in Python.

In [part four](#), you'll learn how to store the model in a database, and then create stored procedures from the Python scripts you developed in parts two and three. The stored procedures will run on the server to make predictions based on new data.

Prerequisites

- SQL Server Machine Learning Services - To install Machine Learning Services, see the [Windows installation guide](#).
- Python IDE - This tutorial uses a Python notebook in [Azure Data Studio](#). For more information, see [How to use notebooks in Azure Data Studio](#).

- SQL query tool - This tutorial assumes you're using [Azure Data Studio](#).
- Additional Python packages - The examples in this tutorial series use the following Python packages that may not be installed by default:
 - pandas
 - pyodbc
 - sklearn

To install these packages:

1. In your Azure Data Studio notebook, select **Manage Packages**.
2. In the **Manage Packages** pane, select the **Add new** tab.
3. For each of the following packages, enter the package name, select **Search**, then select **Install**.

As an alternative, you can open a **Command Prompt**, change to the installation path for the version of Python you use in Azure Data Studio (for example, `cd %LocalAppData%\Programs\Python\Python37-32`), then run `pip install` for each package.

Restore the sample database

The sample database used in this tutorial has been saved to a `.bak` database backup file for you to download and use.

1. Download the file [TutorialDB.bak](#).
2. Follow the directions in [Restore a database from a backup file](#) in Azure Data Studio, using these details:
 - Import from the `TutorialDB.bak` file you downloaded.
 - Name the target database `TutorialDB`.
3. You can verify that the restored database exists by querying the `dbo.rental_data` table:

SQL

```
USE TutorialDB;  
SELECT * FROM [dbo].[rental_data];
```

Clean up resources

If you're not going to continue with this tutorial, delete the `TutorialDB` database.

Next steps

In part one of this tutorial series, you completed these steps:


- Installed the prerequisites
- Import a sample database

To prepare the data from the TutorialDB database, follow part two of this tutorial series:

[Python Tutorial: Prepare data to train a linear regression model](#)



Python Tutorial: Prepare data to train a linear regression model with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In part two of this four-part tutorial series, you'll prepare data from a database using Python. Later in this series, you'll use this data to train and deploy a linear regression model in Python with SQL Server Machine Learning Services.

In this article, you'll learn how to:

-  Load the data from the database into a **pandas** data frame
-  Prepare the data in Python by removing some columns

In [part one](#), you learned how to restore the sample database.

In [part three](#), you'll learn how to train a linear regression machine learning model in Python.

In [part four](#), you'll learn how to store the model in a database, and then create stored procedures from the Python scripts you developed in parts two and three. The stored procedures will run on the server to make predictions based on new data.

Prerequisites

- Part two of this tutorial assumes you have completed [part one](#) and its prerequisites.

Explore and prepare the data

To use the data in Python, you'll load the data from the database into a pandas data frame.

Create a new Python notebook in Azure Data Studio and run the script below.

The Python script below imports the dataset from the `dbo.rental_data` table in your database to a pandas data frame `df`.

In the connection string, replace connection details as needed. To use Windows authentication with an ODBC connection string, specify `Trusted_Connection=Yes;` instead of the `UID` and `PWD` parameters.

Python

```
import pyodbc
import pandas
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Connection string to your SQL Server instance
conn_str = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server}; SERVER=
<server>; DATABASE=TutorialDB;UID=<username>;PWD=<password>')

query_str = 'SELECT Year, Month, Day, Rentalcount, Weekday, Holiday, Snow
FROM dbo.rental_data'

df = pandas.read_sql(sql=query_str, con=conn_str)
print("Data frame:", df)
```

You should see results similar to the following.

results

```
Data frame:      Year  Month  Day  Rentalcount  WeekDay  Holiday  Snow
0    2014     1   20     445           2         1     0
1    2014     2   13      40           5         0     0
2    2013     3   10     456           1         0     0
3    2014     3   31      38           2         0     0
4    2014     4   24      23           5         0     0
..     ...     ...   ...         ...         ...     ...
448  2013     2   19      57           3         0     1
449  2015     3   18      26           4         0     0
450  2015     3   24      29           3         0     1
451  2014     3   26      50           4         0     1
452  2015    12    6     377           1         0     1

[453 rows x 7 columns]
```

Filter the columns from the dataframe to remove ones we don't want to use in the training. `Rentalcount` should not be included as it is the target of the predictions.

Python

```
columns = df.columns.tolist()
columns = [c for c in columns if c not in ["Year", "Rentalcount"]]
```

```
print("Training set:", test[columns])
```

Note the data the training set will have access to:

results

```
Training set:      Month  Day  Weekday  Holiday  Snow
1         2   13      5         0        0
3         3   31      2         0        0
7         3    8      7         0        0
15        3    4      2         0        1
22        1   18      1         0        0
..        ...  ...    ...      ...      ...
416       4   13      1         0        1
421       1   21      3         0        1
438       2   19      4         0        1
441       2    3      3         0        1
447       1    4      6         0        1
```

[91 rows x 5 columns]

Next steps

In part two of this tutorial series, you completed these steps:

- Load the data from the database into a **pandas** data frame
- Prepare the data in Python by removing some columns

To train a machine learning model that uses data from the `TutorialDB` database, follow part three of this tutorial series:

[Python Tutorial: Train a linear regression model](#)



Python tutorial: Train a linear regression model with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In part three of this four-part tutorial series, you'll train a linear regression model in Python. In the next part of this series, you'll deploy this model in a SQL Server database with Machine Learning Services.

In this article, you'll learn how to:

-  Train a linear regression model
-  Make predictions using the linear regression model

In [part one](#), you learned how to restore the sample database.

In [part two](#), you learned how to load the data from a database into a Python data frame, and prepare the data in Python.

In [part four](#), you'll learn how to store the model in a database, and then create stored procedures from the Python scripts you developed in parts two and three. The stored procedures will run in on the server to make predictions based on new data.

Prerequisites

- Part three of this tutorial assumes you have completed [part one](#) and its prerequisites.

Train the model

In order to predict, you have to find a function (model) that best describes the dependency between the variables in our dataset. This called training the model. The training dataset will be a subset of the entire dataset from the pandas data frame `df` that you created in part two of this series.

You will train model `lin_model` using a linear regression algorithm.

```
Python
```

```
# Store the variable we'll be predicting on.  
target = "Rentalcount"
```

```

# Generate the training set. Set random_state to be able to replicate
results.
train = df.sample(frac=0.8, random_state=1)

# Select anything not in the training set and put it in the testing set.
test = df.loc[~df.index.isin(train.index)]

# Print the shapes of both sets.
print("Training set shape:", train.shape)
print("Testing set shape:", test.shape)

# Initialize the model class.
lin_model = LinearRegression()

# Fit the model to the training data.
lin_model.fit(train[columns], train[target])

```

You should see results similar to the following.

results

```

Training set shape: (362, 7)
Testing set shape: (91, 7)

```

Make predictions

Use a predict function to predict the rental counts using the model `lin_model`.

Python

```

# Generate our predictions for the test set.
lin_predictions = lin_model.predict(test[columns])
print("Predictions:", lin_predictions)

# Compute error between our test predictions and the actual values.
lin_mse = mean_squared_error(lin_predictions, test[target])
print("Computed error:", lin_mse)

```

You should see results similar to the following.

results

```

Predictions: [124.41293228 123.8095075 117.67253182 209.39332151
135.46159387
199.50603805 472.14918499 90.15781602 216.61319499 120.30710327
89.47591091 127.71290441 207.44065517 125.68466139 201.38119194
204.29377218 127.4494643 113.42721447 127.37388762 94.66754136

```



```
90.21979191 173.86647615 130.34747586 111.81550069 118.88131715
124.74028405 211.95038051 202.06309706 123.53053083 167.06313191
206.24643852 122.64812937 179.98791527 125.1558454 168.00847713
120.2305587 196.60802649 117.00616326 173.20010759 89.9563518
 92.11048236 120.91052805 175.47818992 129.65196995 120.97443971
175.95863082 127.24800008 135.05866542 206.49627783 91.63004147
115.78280925 208.92841718 213.5137192 212.83278197 96.74415948
 95.1324457 199.9089665 206.10791806 126.16510228 120.0281266
209.08150631 132.88996619 178.84110582 128.85971386 124.67637239
115.58134503 96.82167192 514.61789505 125.48319717 207.50359894
121.64080826 201.9381774 113.22575025 202.46505762 90.7002328
 92.31194658 201.25627228 516.97252195 91.36660136 599.27093251
199.6445585 123.66905128 117.4710676 173.12259514 129.60359486
209.59478573 206.29481361 210.69322009 205.50255751 210.88011563
207.65572019]
```

Computed error: 35003.54030828391

Next steps

In part three of this tutorial series, you completed these steps:



- Train a linear regression model
- Make predictions using the linear regression model

To deploy the machine learning model you've created, follow part four of this tutorial series:

[Python Tutorial: Deploy a machine learning model](#)





Python Tutorial: Deploy a linear regression model with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In part four of this four-part tutorial series, you'll deploy a linear regression model developed in Python into a SQL Server database using Machine Learning Services.

In this article, you'll learn how to:

-  Create a stored procedure that generates the machine learning model
-  Store the model in a database table
-  Create a stored procedure that makes predictions using the model
-  Execute the model with new data

In [part one](#), you learned how to restore the sample database.

In [part two](#), you learned how to load the data from a database into a Python data frame, and prepare the data in Python.

In [part three](#), you learned how to train a linear regression machine learning model in Python.

Prerequisites

- Part four of this tutorial assumes you have completed [part one](#) and its prerequisites.

Create a stored procedure that generates the model

Now, using the Python scripts you developed, create a stored procedure `generate_rental_py_model` that trains and generates the linear regression model using `LinearRegression` from `scikit-learn`.

Run the following T-SQL statement in Azure Data Studio to create the stored procedure to train the model.

SQL

```
-- Stored procedure that trains and generates a Python model using the
rental_data and a linear regression algorithm
DROP PROCEDURE IF EXISTS generate_rental_py_model;
go
CREATE PROCEDURE generate_rental_py_model (@trained_model varbinary(max)
OUTPUT)
AS
BEGIN
    EXECUTE sp_execute_external_script
        @language = N'Python'
        , @script = N'
from sklearn.linear_model import LinearRegression
import pickle

df = rental_train_data

# Get all the columns from the dataframe.
columns = df.columns.tolist()

# Store the variable well be predicting on.
target = "RentalCount"

# Initialize the model class.
lin_model = LinearRegression()

# Fit the model to the training data.
lin_model.fit(df[columns], df[target])

# Before saving the model to the DB table, convert it to a binary object
trained_model = pickle.dumps(lin_model)'
, @input_data_1 = N'select "RentalCount", "Year", "Month", "Day", "WeekDay",
"Snow", "Holiday" from dbo.rental_data where Year < 2015'
, @input_data_1_name = N'rental_train_data'
, @params = N'@trained_model varbinary(max) OUTPUT'
, @trained_model = @trained_model OUTPUT;
END;
GO
```

Store the model in a database table

Create a table in the TutorialDB database and then save the model to the table.

1. Run the following T-SQL statement in Azure Data Studio to create a table called **dbo.rental_py_models** which is used to store the model.

SQL

```

USE TutorialDB;
DROP TABLE IF EXISTS dbo.rental_py_models;
GO
CREATE TABLE dbo.rental_py_models (
    model_name VARCHAR(30) NOT NULL DEFAULT('default model') PRIMARY
KEY,
    model VARBINARY(MAX) NOT NULL
);
GO

```

2. Save the model to the table as a binary object, with the model name **linear_model**.

SQL

```

DECLARE @model VARBINARY(MAX);
EXECUTE generate_rental_py_model @model OUTPUT;

INSERT INTO rental_py_models (model_name, model) VALUES('linear_model',
@model);

```

Create a stored procedure that makes predictions

1. Create a stored procedure **py_predict_rentalcount** that makes predictions using the trained model and a set of new data. Run the T-SQL below in Azure Data Studio.

SQL

```

DROP PROCEDURE IF EXISTS py_predict_rentalcount;
GO
CREATE PROCEDURE py_predict_rentalcount (@model varchar(100))
AS
BEGIN
    DECLARE @py_model varbinary(max) = (select model from
rental_py_models where model_name = @model);

    EXECUTE sp_execute_external_script
        @language = N'Python',
        @script = N'

# Import the scikit-learn function to compute error.
from sklearn.metrics import mean_squared_error
import pickle
import pandas

rental_model = pickle.loads(py_model)

```

```

df = rental_score_data

# Get all the columns from the dataframe.
columns = df.columns.tolist()

# Variable you will be predicting on.
target = "RentalCount"

# Generate the predictions for the test set.
lin_predictions = rental_model.predict(df[columns])
print(lin_predictions)

# Compute error between the test predictions and the actual values.
lin_mse = mean_squared_error(lin_predictions, df[target])
#print(lin_mse)

predictions_df = pandas.DataFrame(lin_predictions)

OutputDataSet = pandas.concat([predictions_df, df["RentalCount"],
df["Month"], df["Day"], df["WeekDay"], df["Snow"], df["Holiday"],
df["Year"]], axis=1)
'
, @input_data_1 = N'Select "RentalCount", "Year" ,"Month", "Day",
"WeekDay", "Snow", "Holiday" from rental_data where Year = 2015'
, @input_data_1_name = N'rental_score_data'
, @params = N'@py_model varbinary(max)'
, @py_model = @py_model
with result sets (("RentalCount_Predicted" float, "RentalCount" float,
"Month" float,"Day" float,"WeekDay" float,"Snow" float,"Holiday" float,
"Year" float));

END;
GO

```

2. Create a table for storing the predictions.

```

SQL

DROP TABLE IF EXISTS [dbo].[py_rental_predictions];
GO

CREATE TABLE [dbo].[py_rental_predictions](
[RentalCount_Predicted] [int] NULL,
[RentalCount_Actual] [int] NULL,
[Month] [int] NULL,
[Day] [int] NULL,
[WeekDay] [int] NULL,
[Snow] [int] NULL,
[Holiday] [int] NULL,
[Year] [int] NULL
) ON [PRIMARY]
GO

```

3. Execute the stored procedure to predict rental counts

SQL

```
--Insert the results of the predictions for test set into a table
INSERT INTO py_rental_predictions
EXEC py_predict_rentalcount 'linear_model';

-- Select contents of the table
SELECT * FROM py_rental_predictions;
```

You should see results similar to the following.

	RentalCount_Predicted	RentalCount_Actual	Month	Day	WeekDay	Snow	Holiday	Year
1	41	42	2	11	4	0	0	2015
2	360	360	3	29	1	0	0	2015
3	19	20	4	22	4	0	0	2015
4	41	42	3	6	6	0	0	2015
5	405	405	2	28	7	1	0	2015
6	37	38	1	12	2	1	0	2015
7	327	327	1	24	7	0	0	2015
8	33	34	4	10	6	0	0	2015
9	36	37	4	16	5	1	0	2015
10	514	514	1	18	1	0	0	2015
11	109	110	12	31	5	0	0	2015

You have successfully created, trained, and deployed a model. You then used that model in a stored procedure to predict values based on new data.

Next steps

In part four of this tutorial series, you completed these steps:

- Create a stored procedure that generates the machine learning model
- Store the model in a database table
- Create a stored procedure that makes predictions using the model
- Execute the model with new data

To learn more about using Python with SQL machine learning, see:

- [Python tutorials](#)

Python tutorial: Categorizing customers using k-means clustering with SQL machine learning

Article • 04/17/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In this four-part tutorial series, use Python to develop and deploy a K-Means clustering model in [SQL Server Machine Learning Services](#) to cluster customer data.

In part one of this series, set up the prerequisites for the tutorial and then restore a sample dataset to a database. Later in this series, use this data to train and deploy a clustering model in Python with SQL machine learning.

In parts two and three of this series, develop some Python scripts in an Azure Data Studio notebook to analyze and prepare your data and train a machine learning model. Then, in part four, run those Python scripts inside a database using stored procedures.

Clustering can be explained as organizing data into groups where members of a group are similar in some way. For this tutorial series, imagine you own a retail business. Use the **K-Means** algorithm to perform the clustering of customers in a dataset of product purchases and returns. By clustering customers, you can focus your marketing efforts more effectively by targeting specific groups. K-Means clustering is an *unsupervised learning* algorithm that looks for patterns in data based on similarities.

In this article, learn how to:

-  Restore a sample database

In [part two](#), learn how to prepare the data from a database to perform clustering.

In [part three](#), learn how to create and train a K-Means clustering model in Python.

In [part four](#), learn how to create a stored procedure in a database that can perform clustering in Python based on new data.

Prerequisites

- [SQL Server Machine Learning Services](#) with the Python language option - Follow the installation instructions in the [Windows installation guide](#).

- [Azure Data Studio](#). use a notebook in Azure Data Studio for both Python and SQL. For more information about notebooks, see [How to use notebooks in Azure Data Studio](#).
- Additional Python packages - The examples in this tutorial series use Python packages that you may or may not have installed.

Open an **Administrative Command Prompt** and change to the installation path for the version of Python you use in Azure Data Studio. For example, `cd %LocalAppData%\Programs\Python\Python37-32`. Then run the following commands to install any of these packages that aren't already installed. Ensure these packages are installed in the correct Python installation location. You can use the option `-t` to specify the destination directory.

Console

```
pip install matplotlib
pip install pandas
pip install pyodbc
pip install scipy
pip install scikit-learn
```

Restore the sample database

The sample dataset used in this tutorial has been saved to a **.bak** database backup file for you to download and use. This dataset is derived from the [tpcx-bb](#) dataset provided by the [Transaction Processing Performance Council \(TPC\)](#).

1. Download the file [tpcxbb_1gb.bak](#).
2. Follow the directions in [Restore a database from a backup file](#) in Azure Data Studio, using these details:
 - Import from the **tpcxbb_1gb.bak** file you downloaded
 - Name the target database "tpcxbb_1gb"
3. You can verify that the dataset exists after you have restored the database by querying the **dbo.customer** table:

SQL

```
USE tpcxbb_1gb;
SELECT * FROM [dbo].[customer];
```


Clean up resources

If you're not going to continue with this tutorial, delete the tpcxbb_1gb database.

Next steps

In part one of this tutorial series, you completed these steps:



- Restore a sample database

To prepare the data for the machine learning model, follow part two of this tutorial series:

[Python tutorial: Prepare data to perform clustering](#)



Python tutorial: Prepare data to categorize customers with SQL machine learning

Article • 04/17/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In part two of this four-part tutorial series, you'll restore and prepare the data from a database using Python. Later in this series, you'll use this data to train and deploy a clustering model in Python with SQL Server Machine Learning Services.

In this article, you'll learn how to:

-  Separate customers along different dimensions using Python
-  Load the data from the database into a Python data frame

In [part one](#), you installed the prerequisites and restored the sample database.

In [part three](#), you'll learn how to create and train a K-Means clustering model in Python.

In [part four](#), you'll learn how to create a stored procedure in a database that can perform clustering in Python based on new data.

Prerequisites

- Part two of this tutorial assumes you have fulfilled the prerequisites of [part one](#).

Separate customers

To prepare for clustering customers, you'll first separate customers along the following dimensions:

- **orderRatio** = return order ratio (total number of orders partially or fully returned versus the total number of orders)
- **itemsRatio** = return item ratio (total number of items returned versus the number of items purchased)
- **monetaryRatio** = return amount ratio (total monetary amount of items returned versus the amount purchased)
- **frequency** = return frequency

Open a new notebook in Azure Data Studio and enter the following script.

In the connection string, replace connection details as needed.

Python

```
# Load packages.
import pyodbc
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.spatial import distance as sci_distance
from sklearn import cluster as sk_cluster

#####
#####

## Connect to DB and select data

#####
#####

# Connection string to connect to SQL Server named instance.
conn_str = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server}; SERVER=
<server>; DATABASE=tpcxbb_1gb; UID=<username>; PWD=<password>')

input_query = '''SELECT
ss_customer_sk AS customer,
ROUND(COALESCE(returns_count / NULLIF(1.0*orders_count, 0), 0), 7) AS
orderRatio,
ROUND(COALESCE(returns_items / NULLIF(1.0*orders_items, 0), 0), 7) AS
itemsRatio,
ROUND(COALESCE(returns_money / NULLIF(1.0*orders_money, 0), 0), 7) AS
monetaryRatio,
COALESCE(returns_count, 0) AS frequency
FROM
(
    SELECT
        ss_customer_sk,
        -- return order ratio
        COUNT(distinct(ss_ticket_number)) AS orders_count,
        -- return ss_item_sk ratio
        COUNT(ss_item_sk) AS orders_items,
        -- return monetary amount ratio
        SUM( ss_net_paid ) AS orders_money
    FROM store_sales s
    GROUP BY ss_customer_sk
) orders
LEFT OUTER JOIN
(
    SELECT
        sr_customer_sk,
        -- return order ratio
        count(distinct(sr_ticket_number)) as returns_count,
```

```

-- return ss_item_sk ratio
COUNT(sr_item_sk) as returns_items,
-- return monetary amount ratio
SUM( sr_return_amt ) AS returns_money
FROM store_returns
GROUP BY sr_customer_sk ) returned ON ss_customer_sk=sr_customer_sk'''

# Define the columns we wish to import.
column_info = {
    "customer": {"type": "integer"},
    "orderRatio": {"type": "integer"},
    "itemsRatio": {"type": "integer"},
    "frequency": {"type": "integer"}
}

```

Load the data into a data frame

Results from the query are returned to Python using the Pandas `read_sql` function. As part of the process, you'll use the column information you defined in the previous script.

Python

```
customer_data = pd.read_sql(input_query, conn_str)
```

Now display the beginning of the data frame to verify it looks correct.

Python

```
print("Data frame:", customer_data.head(n=5))
```

results

```
Rows Read: 37336, Total Rows Processed: 37336, Total Chunk Time: 0.172
seconds
```

Data frame:	customer	orderRatio	itemsRatio	monetaryRatio	frequency
0	29727.0	0.000000	0.000000	0.000000	0
1	97643.0	0.068182	0.078176	0.037034	3
2	57247.0	0.000000	0.000000	0.000000	0
3	32549.0	0.086957	0.068657	0.031281	4
4	2040.0	0.000000	0.000000	0.000000	0

Clean up resources

If you're not going to continue with this tutorial, delete the `tpcxbb_1gb` database.

Next steps

In part two of this tutorial series, you completed these steps:

- Separate customers along different dimensions using Python
- Load the data from the database into a Python data frame

To create a machine learning model that uses this customer data, follow part three of this tutorial series:

[Python tutorial: Create a predictive model](#)




Python tutorial: Build a model to categorize customers with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In part three of this four-part tutorial series, you'll build a K-Means model in Python to perform clustering. In the next part of this series, you'll deploy this model in a database with SQL Server Machine Learning Services.

In this article, you'll learn how to:

-  Define the number of clusters for a K-Means algorithm
-  Perform clustering
-  Analyze the results

In [part one](#), you installed the prerequisites and restored the sample database.

In [part two](#), you learned how to prepare the data from a database to perform clustering.

In [part four](#), you'll learn how to create a stored procedure in a database that can perform clustering in Python based on new data.

Prerequisites

- Part three of this tutorial assumes you have fulfilled the prerequisites of [part one](#), and completed the steps in [part two](#).

Define the number of clusters

To cluster your customer data, you'll use the **K-Means** clustering algorithm, one of the simplest and most well-known ways of grouping data. You can read more about K-Means in [A complete guide to K-means clustering algorithm](#)^[2].

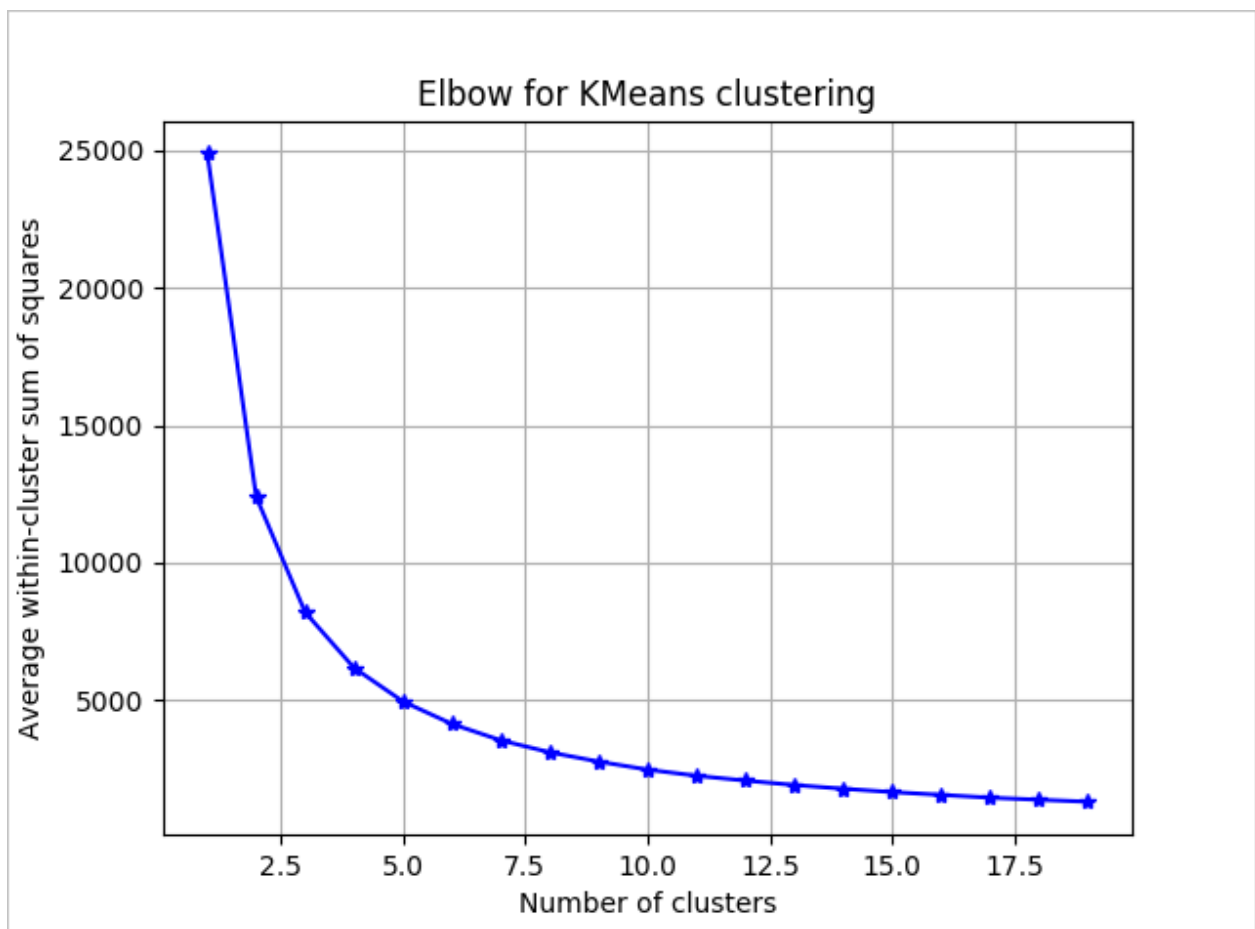
The algorithm accepts two inputs: The data itself, and a predefined number "k" representing the number of clusters to generate. The output is k clusters with the input data partitioned among the clusters.

The goal of K-means is to group the items into k clusters such that all items in same cluster are as similar to each other, and as different from items in other clusters, as possible.

To determine the number of clusters for the algorithm to use, use a plot of the within groups sum of squares, by number of clusters extracted. The appropriate number of clusters to use is at the bend or "elbow" of the plot.

Python

```
#####  
#####  
## Determine number of clusters using the Elbow method  
#####  
#####  
  
cdata = customer_data  
K = range(1, 20)  
KM = (sk_cluster.KMeans(n_clusters=k).fit(cdata) for k in K)  
centroids = (k.cluster_centers_ for k in KM)  
  
D_k = (sci_distance.cdist(cdata, cent, 'euclidean') for cent in centroids)  
dist = (np.min(D, axis=1) for D in D_k)  
avgWithinSS = [sum(d) / cdata.shape[0] for d in dist]  
plt.plot(K, avgWithinSS, 'b*-')  
plt.grid(True)  
plt.xlabel('Number of clusters')  
plt.ylabel('Average within-cluster sum of squares')  
plt.title('Elbow for KMeans clustering')  
plt.show()
```



Based on the graph, it looks like $k = 4$ would be a good value to try. That k value will group the customers into four clusters.

Perform clustering

In the following Python script, you'll use the KMeans function from the sklearn package.

Python

```
#####
#####
## Perform clustering using Kmeans
#####
#####

# It looks like k=4 is a good number to use based on the elbow graph.
n_clusters = 4

means_cluster = sk_cluster.KMeans(n_clusters=n_clusters, random_state=111)
columns = ["orderRatio", "itemsRatio", "monetaryRatio", "frequency"]
est = means_cluster.fit(customer_data[columns])
clusters = est.labels_
customer_data['cluster'] = clusters

# Print some data about the clusters:
```



```
# For each cluster, count the members.
for c in range(n_clusters):
    cluster_members=customer_data[customer_data['cluster'] == c][:]
    print('Cluster{}(n={})'.format(c, len(cluster_members)))
    print('-'* 17)
print(customer_data.groupby(['cluster']).mean())
```

Analyze the results

Now that you've performed clustering using K-Means, the next step is to analyze the result and see if you can find any actionable information.

Look at the clustering mean values and cluster sizes printed from the previous script.

results

Cluster0(n=31675):

Cluster1(n=4989):

Cluster2(n=1):

Cluster3(n=671):

	customer	orderRatio	itemsRatio	monetaryRatio	frequency
cluster					
0	50854.809882	0.000000	0.000000	0.000000	0.000000
1	51332.535779	0.721604	0.453365	0.307721	1.097815
2	57044.000000	1.000000	2.000000	108.719154	1.000000
3	48516.023845	0.136277	0.078346	0.044497	4.271237

The four cluster means are given using the variables defined in [part one](#):

- *orderRatio* = return order ratio (total number of orders partially or fully returned versus the total number of orders)
- *itemsRatio* = return item ratio (total number of items returned versus the number of items purchased)
- *monetaryRatio* = return amount ratio (total monetary amount of items returned versus the amount purchased)
- *frequency* = return frequency

Data mining using K-Means often requires further analysis of the results, and further steps to better understand each cluster, but it can provide some good leads. Here are a couple ways you could interpret these results:

- Cluster 0 seems to be a group of customers that are not active (all values are zero).
- Cluster 3 seems to be a group that stands out in terms of return behavior.

Cluster 0 is a set of customers who are clearly not active. Perhaps you can target marketing efforts towards this group to trigger an interest for purchases. In the next step, you'll query the database for the email addresses of customers in cluster 0, so that you can send a marketing email to them.

Clean up resources

If you're not going to continue with this tutorial, delete the `tpcxbb_1gb` database.

Next steps

In part three of this tutorial series, you completed these steps:



- Define the number of clusters for a K-Means algorithm
- Perform clustering
- Analyze the results

To deploy the machine learning model you've created, follow part four of this tutorial series:

[Python tutorial: Deploy a clustering model](#)

Python tutorial: Deploy a model to categorize customers with SQL machine learning

Article • 04/17/2023




Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In part four of this four-part tutorial series, you'll deploy a clustering model, developed in Python, into a database using SQL Server Machine Learning Services.

In order to perform clustering on a regular basis, as new customers are registering, you need to be able call the Python script from any App. To do that, you can deploy the Python script in a database by putting the Python script inside a SQL stored procedure. Because your model executes in the database, it can easily be trained against data stored in the database.

In this section, you'll move the Python code you just wrote onto the server and deploy clustering.

In this article, you'll learn how to:

-  Create a stored procedure that generates the model
-  Perform clustering on the server
-  Use the clustering information

In [part one](#), you installed the prerequisites and restored the sample database.

In [part two](#), you learned how to prepare the data from a database to perform clustering.

In [part three](#), you learned how to create and train a K-Means clustering model in Python.

Prerequisites

- Part four of this tutorial series assumes you have fulfilled the prerequisites of [part one](#), and completed the steps in [part two](#) and [part three](#).

Create a stored procedure that generates the model

Run the following T-SQL script to create the stored procedure. The procedure recreates the steps you developed in parts one and two of this tutorial series:

- classify customers based on their purchase and return history
- generate four clusters of customers using a K-Means algorithm

SQL

```
USE [tpcxbb_1gb]
GO

DROP procedure IF EXISTS [dbo].[py_generate_customer_return_clusters];
GO

CREATE procedure [dbo].[py_generate_customer_return_clusters]
AS

BEGIN
    DECLARE

    -- Input query to generate the purchase history & return metrics
    @input_query NVARCHAR(MAX) = N'
SELECT
    ss_customer_sk AS customer,
    CAST( (ROUND(COALESCE(returns_count / NULLIF(1.0*orders_count, 0), 0), 0), 7)
) AS FLOAT) AS orderRatio,
    CAST( (ROUND(COALESCE(returns_items / NULLIF(1.0*orders_items, 0), 0), 0), 7)
) AS FLOAT) AS itemsRatio,
    CAST( (ROUND(COALESCE(returns_money / NULLIF(1.0*orders_money, 0), 0), 0), 7)
) AS FLOAT) AS monetaryRatio,
    CAST( (COALESCE(returns_count, 0)) AS FLOAT) AS frequency
FROM
    (
        SELECT
            ss_customer_sk,
            -- return order ratio
            COUNT(distinct(ss_ticket_number)) AS orders_count,
            -- return ss_item_sk ratio
            COUNT(ss_item_sk) AS orders_items,
            -- return monetary amount ratio
            SUM( ss_net_paid ) AS orders_money
        FROM store_sales s
        GROUP BY ss_customer_sk
    ) orders
LEFT OUTER JOIN
    (
        SELECT
            sr_customer_sk,
            -- return order ratio
            count(distinct(sr_ticket_number)) as returns_count,
            -- return ss_item_sk ratio
            COUNT(sr_item_sk) as returns_items,
            -- return monetary amount ratio
```

```

        SUM( sr_return_amt ) AS returns_money
    FROM store_returns
    GROUP BY sr_customer_sk
) returned ON ss_customer_sk=sr_customer_sk
'

EXEC sp_execute_external_script
    @language = N'Python'
    , @script = N'

import pandas as pd
from sklearn.cluster import KMeans

#get data from input query
customer_data = my_input_data

#We concluded in step 2 in the tutorial that 4 would be a good number of
clusters
n_clusters = 4

#Perform clustering
est = KMeans(n_clusters=n_clusters,
random_state=111).fit(customer_data[["orderRatio", "itemsRatio", "monetaryRati
o", "frequency"]])
clusters = est.labels_
customer_data["cluster"] = clusters

OutputDataSet = customer_data
'

    , @input_data_1 = @input_query
    , @input_data_1_name = N'my_input_data'
        with result sets (("Customer" int, "orderRatio"
float,"itemsRatio" float,"monetaryRatio" float,"frequency" float,"cluster"
float));
END;
GO

```

Perform clustering

Now that you've created the stored procedure, execute the following script to perform clustering using the procedure.

SQL

```

--Create a table to store the predictions in

DROP TABLE IF EXISTS [dbo].[py_customer_clusters];
GO

CREATE TABLE [dbo].[py_customer_clusters] (
    [Customer] [bigint] NULL

```

```

, [OrderRatio] [float] NULL
, [itemsRatio] [float] NULL
, [monetaryRatio] [float] NULL
, [frequency] [float] NULL
, [cluster] [int] NULL
,
) ON [PRIMARY]
GO

--Execute the clustering and insert results into table
INSERT INTO py_customer_clusters
EXEC [dbo].[py_generate_customer_return_clusters];

-- Select contents of the table to verify it works
SELECT * FROM py_customer_clusters;

```

Use the clustering information

Because you stored the clustering procedure in the database, it can perform clustering efficiently against customer data stored in the same database. You can execute the procedure whenever your customer data is updated and use the updated clustering information.

Suppose you want to send a promotional email to customers in cluster 0, the group that was inactive (you can see how the four clusters were described in [part three](#) of this tutorial). The following code selects the email addresses of customers in cluster 0.

```

SQL

USE [tpcxbb_1gb]
--Get email addresses of customers in cluster 0 for a promotion campaign
SELECT customer.[c_email_address], customer.c_customer_sk
FROM dbo.customer
JOIN
[dbo].[py_customer_clusters] as c
ON c.Customer = customer.c_customer_sk
WHERE c.cluster = 0

```

You can change the `c.cluster` value to return email addresses for customers in other clusters.

Clean up resources

When you're finished with this tutorial, you can delete the `tpcxbb_1gb` database.

Next steps

In part four of this tutorial series, you completed these steps:

- Create a stored procedure that generates the model
- Perform clustering on the server
- Use the clustering information

To learn more about using Python in SQL machine learning, see:

- [Quickstart: Create and run simple Python scripts](#)
- [Other Python tutorials for SQL machine learning](#)
- [Install Python packages with sqlmlutils](#)

Python tutorial: Predict NYC taxi fares with binary classification

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)



In this five-part tutorial series for SQL programmers, you'll learn about Python integration in [SQL Server Machine Learning Services](#).

You'll build and deploy a Python-based machine learning solution using a sample database on SQL Server. You'll use T-SQL, Azure Data Studio or SQL Server Management Studio, and a database instance with SQL machine learning and Python language support.

This tutorial series introduces you to Python functions used in a data modeling workflow. Parts include data exploration, building and training a binary classification model, and model deployment. You'll use sample data from the New York City Taxi and Limousine Commission. The model you'll build predicts whether a trip is likely to result in a tip based on the time of day, distance traveled, and pick-up location.

In the first part of this series, you'll install the prerequisites and restore the sample database. In parts two and three, you'll develop some Python scripts to prepare your data and train a machine learning model. Then, in parts four and five, you'll run those Python scripts inside the database using T-SQL stored procedures.

In this article, you'll:

-  Install prerequisites
-  Restore the sample database

In [part two](#), you'll explore the sample data and generate some plots.

In [part three](#), you'll learn how to create features from raw data by using a Transact-SQL function. You'll then call that function from a stored procedure to create a table that contains the feature values.

In [part four](#), you'll load the modules and call the necessary functions to create and train the model using a SQL Server stored procedure.

In [part five](#), you'll learn how to operationalize the models that you trained and saved in part four.

ⓘ Note

This tutorial is available in both R and Python. For the R version, see [R tutorial: Predict NYC taxi fares with binary classification](#).

Prerequisites

- Install [SQL Server Machine Learning Services with Python](#)
- [Grant permissions to execute Python scripts](#)
- Restore the [NYC Taxi demo database](#)

All tasks can be done using Transact-SQL stored procedures in Azure Data Studio or Management Studio.

This tutorial series assumes familiarity with basic database operations such as creating databases and tables, importing data, and writing SQL queries. It does not assume you know Python and all Python code is provided.

Background for SQL developers

The process of building a machine learning solution is a complex one that can involve multiple tools, and the coordination of subject matter experts across several phases:

- obtaining and cleaning data
- exploring the data and building features useful for modeling
- training and tuning the model
- deployment to production

Development and testing of the actual code is best performed using a dedicated development environment. However, after the script is fully tested, you can easily deploy it to SQL Server using Transact-SQL stored procedures in the familiar environment of Azure Data Studio or Management Studio. Wrapping external code in stored procedures is the primary mechanism for operationalizing code in SQL Server.

After the model has been saved to the database, you can call the model for predictions from Transact-SQL by using stored procedures.

Whether you're a SQL programmer new to Python, or a Python developer new to SQL, this five-part tutorial series introduces a typical workflow for conducting in-database analytics with Python and SQL Server.

Next steps


In this article, you:

- ✓ Installed prerequisites
- ✓ Restored the sample database

[Python tutorial: Explore and visualize data](#)



Python tutorial: Explore and visualize data

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In part two of this five-part tutorial series, you'll explore the sample data and generate some plots. Later, you'll learn how to serialize graphics objects in Python, and then deserialize those objects and make plots.

In this article, you'll:

-  Review the sample data
-  Create plots using Python in T-SQL

In [part one](#), you installed the prerequisites and restored the sample database.

In [part three](#), you'll learn how to create features from raw data by using a Transact-SQL function. You'll then call that function from a stored procedure to create a table that contains the feature values.

In [part four](#), you'll load the modules and call the necessary functions to create and train the model using a SQL Server stored procedure.

In [part five](#), you'll learn how to operationalize the models that you trained and saved in part four.

Review the data

First, take a minute to browse the data schema, as we've made some changes to make it easier to use the NYC Taxi data

- The original dataset used separate files for the taxi identifiers and trip records. We've joined the two original datasets on the columns *medallion*, *hack_license*, and *pickup_datetime*.
- The original dataset spanned many files and was quite large. We've downsampled to get just 1% of the original number of records. The current data table has 1,703,957 rows and 23 columns.

Taxi identifiers

The *medallion* column represents the taxi's unique ID number.

The *hack_license* column contains the taxi driver's license number (anonymized).

Trip and fare records

Each trip record includes the pickup and drop-off location and time, and the trip distance.

Each fare record includes payment information such as the payment type, total amount of payment, and the tip amount.

The last three columns can be used for various machine learning tasks. The *tip_amount* column contains continuous numeric values and can be used as the **label** column for regression analysis. The *tipped* column has only yes/no values and is used for binary classification. The *tip_class* column has multiple **class labels** and therefore can be used as the label for multi-class classification tasks.

The values used for the label columns are all based on the `tip_amount` column, using these business rules:

- Label column `tipped` has possible values 0 and 1
If `tip_amount > 0`, `tipped = 1`; otherwise `tipped = 0`
- Label column `tip_class` has possible class values 0-4
Class 0: `tip_amount = $0`
Class 1: `tip_amount > $0` and `tip_amount <= $5`
Class 2: `tip_amount > $5` and `tip_amount <= $10`
Class 3: `tip_amount > $10` and `tip_amount <= $20`
Class 4: `tip_amount > $20`

Create plots using Python in T-SQL

Developing a data science solution usually includes intensive data exploration and data visualization. Because visualization is such a powerful tool for understanding the distribution of the data and outliers, Python provides many packages for visualizing data. The **matplotlib** module is one of the more popular libraries for visualization, and includes many functions for creating histograms, scatter plots, box plots, and other data exploration graphs.

In this section, you learn how to work with plots using stored procedures. Rather than open the image on the server, you store the Python object `plot` as **varbinary** data, and then write that to a file that can be shared or viewed elsewhere.

Create a plot as varbinary data

The stored procedure returns a serialized Python `figure` object as a stream of **varbinary** data. You cannot view the binary data directly, but you can use Python code on the client to deserialize and view the figures, and then save the image file on a client computer.

1. Create the stored procedure `PyPlotMatplotlib`.

In the following script:

- The variable `@query` defines the query text `SELECT tipped FROM nyctaxi_sample`, which is passed to the Python code block as the argument to the script input variable, `@input_data_1`.
- The Python script is fairly simple: `matplotlib` `figure` objects are used to make the histogram and scatter plot, and these objects are then serialized using the `pickle` library.
- The Python graphics object is serialized to a `pandas` DataFrame for output.

SQL

```
DROP PROCEDURE IF EXISTS PyPlotMatplotlib;
GO

CREATE PROCEDURE [dbo].[PyPlotMatplotlib]
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @query nvarchar(max) =
        N'SELECT cast(tipped as int) as tipped, tip_amount, fare_amount
        FROM [dbo].[nyctaxi_sample]'
    EXECUTE sp_execute_external_script
        @language = N'Python',
        @script = N'
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import pandas as pd
import pickle

fig_handle = plt.figure()
plt.hist(InputDataSet.tipped)
plt.xlabel("Tipped")
```

```

plt.ylabel("Counts")
plt.title("Histogram, Tipped")
plot0 = pd.DataFrame(data =[pickle.dumps(fig_handle)], columns =
["plot"])
plt.clf()

plt.hist(InputDataSet.tip_amount)
plt.xlabel("Tip amount ($)")
plt.ylabel("Counts")
plt.title("Histogram, Tip amount")
plot1 = pd.DataFrame(data =[pickle.dumps(fig_handle)], columns =
["plot"])
plt.clf()

plt.hist(InputDataSet.fare_amount)
plt.xlabel("Fare amount ($)")
plt.ylabel("Counts")
plt.title("Histogram, Fare amount")
plot2 = pd.DataFrame(data =[pickle.dumps(fig_handle)], columns =
["plot"])
plt.clf()

plt.scatter( InputDataSet.fare_amount, InputDataSet.tip_amount)
plt.xlabel("Fare Amount ($)")
plt.ylabel("Tip Amount ($)")
plt.title("Tip amount by Fare amount")
plot3 = pd.DataFrame(data =[pickle.dumps(fig_handle)], columns =
["plot"])
plt.clf()

OutputDataSet = plot0.append(plot1, ignore_index=True).append(plot2,
ignore_index=True).append(plot3, ignore_index=True)
',
@input_data_1 = @query
WITH RESULT SETS ((plot varbinary(max)))
END
GO

```

2. Now run the stored procedure with no arguments to generate a plot from the data hard-coded as the input query.

```
SQL
```

```
EXEC [dbo].[PyPlotMatplotlib]
```

3. The results should be something like this:

```
SQL
```

```
plot
0xFFD8FFFE000104A4649...
```

```
0xFFD8FFFE000104A4649...
0xFFD8FFFE000104A4649...
0xFFD8FFFE000104A4649...
```

4. From a [Python client](#), you can now connect to the SQL Server instance that generated the binary plot objects, and view the plots.

To do this, run the following Python code, replacing the server name, database name, and credentials as appropriate (for Windows authentication, replace the `UID` and `PWD` parameters with `Trusted_Connection=True`). Make sure the Python version is the same on the client and the server. Also make sure that the Python libraries on your client (such as matplotlib) are the same or higher version relative to the libraries installed on the server. To view a list of installed packages and their versions, see [Get Python package information](#).

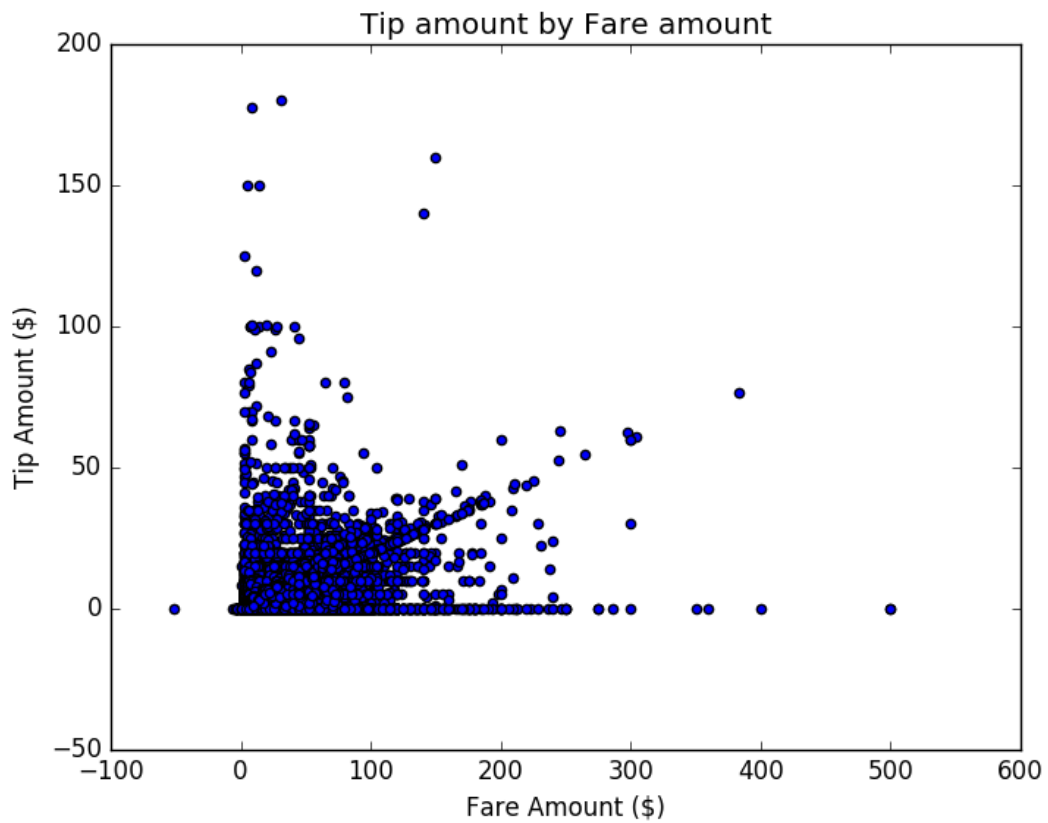
Python

```
%matplotlib notebook
import pyodbc
import pickle
import os
cnxn = pyodbc.connect('DRIVER=SQL Server;SERVER={SERVER_NAME};DATABASE={DB_NAME};UID={USER_NAME};PWD={PASSWORD}')
cursor = cnxn.cursor()
cursor.execute("EXECUTE [dbo].[PyPlotMatplotlib]")
tables = cursor.fetchall()
for i in range(0, len(tables)):
    fig = pickle.loads(tables[i][0])
    fig.savefig(str(i)+'.png')
print("The plots are saved in directory: ",os.getcwd())
```

5. If the connection is successful, you should see a message like the following:

The plots are saved in directory: xxxx

6. The output file is created in the Python working directory. To view the plot, locate the Python working directory, and open the file. The following image shows a plot saved on the client computer.



Next steps

In this article, you:

- ✓ Reviewed the sample data
- ✓ Created plots using Python in T-SQL

[Python tutorial: Create Data Features using T-SQL](#)

Python tutorial: Create Data Features using T-SQL



Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In part three of this five-part tutorial series, you'll learn how to create features from raw data by using a Transact-SQL function. You'll then call that function from a SQL stored procedure to create a table that contains the feature values.

The process of *feature engineering*, creating features from the raw data, can be a critical step in advanced analytics modeling.

In this article, you'll:

-  Modify a custom function to calculate trip distance
-  Save the features using another custom function


In [part one](#), you installed the prerequisites and restored the sample database.

In [part two](#), you explored the sample data and generated some plots.

In [part four](#), you'll load the modules and call the necessary functions to create and train the model using a SQL Server stored procedure.

In [part five](#), you'll learn how to operationalize the models that you trained and saved in part four.

Define the Function

The distance values reported in the original data are based on the reported meter distance, and don't necessarily represent geographical distance or distance traveled. Therefore, you'll need to calculate the direct distance between the pick-up and drop-off points, by using the coordinates available in the source NYC Taxi dataset. You can do this by using the [Haversine formula](#)  in a custom Transact-SQL function.

You'll use one custom T-SQL function, *fnCalculateDistance*, to compute the distance using the Haversine formula, and use a second custom T-SQL function, *fnEngineerFeatures*, to create a table containing all the features.

Calculate trip distance using *fnCalculateDistance*

The function *fnCalculateDistance* is included in the sample database. Take a minute to review the code:

1. In Management Studio, expand **Programmability**, expand **Functions** and then **Scalar-valued functions**.
2. Right-click *fnCalculateDistance*, and select **Modify** to open the Transact-SQL script in a new query window.

It should look something like this:

```
SQL

CREATE FUNCTION [dbo].[fnCalculateDistance] (@Lat1 float, @Long1 float,
@Lat2 float, @Long2 float)
-- User-defined function that calculates the direct distance between
two geographical coordinates
RETURNS float
AS
BEGIN
    DECLARE @distance decimal(28, 10)
    -- Convert to radians
    SET @Lat1 = @Lat1 / 57.2958
    SET @Long1 = @Long1 / 57.2958
    SET @Lat2 = @Lat2 / 57.2958
    SET @Long2 = @Long2 / 57.2958
    -- Calculate distance
    SET @distance = (SIN(@Lat1) * SIN(@Lat2)) + (COS(@Lat1) * COS(@Lat2)
* COS(@Long2 - @Long1))
    --Convert to miles
    IF @distance <> 0
    BEGIN
        SET @distance = 3958.75 * ATAN(SQRT(1 - POWER(@distance, 2)) /
@distance);
    END
    RETURN @distance
END
GO
```

Notes:

- The function is a scalar-valued function, returning a single data value of a predefined type.
- The function takes latitude and longitude values as inputs, obtained from trip pick-up and drop-off locations. The Haversine formula converts locations to radians and uses those values to compute the direct distance in miles between those two locations.

Save the features using *fnEngineerFeatures*

To add the computed value to a table that can be used for training the model, you'll use the custom T-SQL function, *fnEngineerFeatures*. This function is a table-valued function that takes multiple columns as inputs, and outputs a table with multiple feature columns. The purpose of this function is to create a feature set for use in building a model. The function *fnEngineerFeatures* calls the previously created T-SQL function, *fnCalculateDistance*, to get the direct distance between pickup and dropoff locations.

Take a minute to review the code:

SQL

```
CREATE FUNCTION [dbo].[fnEngineerFeatures] (
    @passenger_count int = 0,
    @trip_distance float = 0,
    @trip_time_in_secs int = 0,
    @pickup_latitude float = 0,
    @pickup_longitude float = 0,
    @dropoff_latitude float = 0,
    @dropoff_longitude float = 0)
RETURNS TABLE
AS
    RETURN
    (
        -- Add the SELECT statement with parameter references here
        SELECT
            @passenger_count AS passenger_count,
            @trip_distance AS trip_distance,
            @trip_time_in_secs AS trip_time_in_secs,
            [dbo].[fnCalculateDistance](@pickup_latitude, @pickup_longitude,
            @dropoff_latitude, @dropoff_longitude) AS direct_distance
        )
GO
```

To verify that this function works, you can use it to calculate the geographical distance for those trips where the metered distance was 0 but the pick-up and drop-off locations were different.

SQL

```
SELECT tipped, fare_amount, passenger_count, (trip_time_in_secs/60) as
TripMinutes,
    trip_distance, pickup_datetime, dropoff_datetime,
    dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
    dropoff_latitude, dropoff_longitude) AS direct_distance
FROM nyctaxi_sample
WHERE pickup_longitude != dropoff_longitude and pickup_latitude !=
```

```
dropoff_latitude and trip_distance = 0  
ORDER BY trip_time_in_secs DESC
```

As you can see, the distance reported by the meter doesn't always correspond to geographical distance. This is why feature engineering is important.

In the next part, you'll learn how to use these data features to create and train a machine learning model using Python.

Next steps

In this article, you:

- ✓ Modified a custom function to calculate trip distance
- ✓ Saved the features using another custom function

[Python tutorial: Train and save a Python model using T-SQL](#)

Python tutorial: Train and save a Python model using T-SQL



Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In part four of this five-part tutorial series, you'll learn how to train a machine learning model using the Python packages **scikit-learn** and **revoscalepy**. These Python libraries are already installed with SQL Server machine learning.

You'll load the modules and call the necessary functions to create and train the model using a SQL Server stored procedure. The model requires the data features you engineered in earlier parts of this tutorial series. Finally, you'll save the trained model to a SQL Server table.

In this article, you'll:

-  Create and train a model using a SQL stored procedure
-  Save the trained model to a SQL table

In [part one](#), you installed the prerequisites and restored the sample database.

In [part two](#), you explored the sample data and generated some plots.

In [part three](#), you learned how to create features from raw data by using a Transact-SQL function. You then called that function from a stored procedure to create a table that contains the feature values.

In [part five](#), you'll learn how to operationalize the models that you trained and saved in part four.

Split the sample data into training and testing sets

1. Create a stored procedure called **PyTrainTestSplit** to divide the data in the `nyctaxi_sample` table into two parts: `nyctaxi_sample_training` and `nyctaxi_sample_testing`.

Run the following code to create it:

```
SQL
```

```

DROP PROCEDURE IF EXISTS PyTrainTestSplit;
GO

CREATE PROCEDURE [dbo].[PyTrainTestSplit] (@pct int)
AS

DROP TABLE IF EXISTS dbo.nyctaxi_sample_training
SELECT * into nyctaxi_sample_training FROM nyctaxi_sample WHERE
(ABS(CAST(BINARY_CHECKSUM(medallion,hack_license) as int)) % 100) <
@pct

DROP TABLE IF EXISTS dbo.nyctaxi_sample_testing
SELECT * into nyctaxi_sample_testing FROM nyctaxi_sample
WHERE (ABS(CAST(BINARY_CHECKSUM(medallion,hack_license) as int)) %
100) > @pct
GO

```

2. To divide your data using a custom split, run the stored procedure, and provide an integer parameter that represents the percentage of data to allocate to the training set. For example, the following statement would allocate 60% of data to the training set.

SQL

```

EXEC PyTrainTestSplit 60
GO

```

Build a logistic regression model

After the data has been prepared, you can use it to train a model. You do this by calling a stored procedure that runs some Python code, taking as input the training data table. For this tutorial, you create two models, both binary classification models:

- The stored procedure **PyTrainScikit** creates a tip prediction model using the **scikit-learn** package.
- The stored procedure **TrainTipPredictionModelRxPy** creates a tip prediction model using the **revoscalepy** package.

Each stored procedure uses the input data you provide to create and train a logistic regression model. All Python code is wrapped in the system stored procedure, [sp_execute_external_script](#).

To make it easier to retrain the model on new data, you wrap the call to `sp_execute_external_script` in another stored procedure, and pass in the new training data as a parameter. This section will walk you through that process.

PyTrainScikit

1. In Management Studio, open a new **Query** window and run the following statement to create the stored procedure **PyTrainScikit**. The stored procedure contains a definition of the input data, so you don't need to provide an input query.

```
SQL

DROP PROCEDURE IF EXISTS PyTrainScikit;
GO

CREATE PROCEDURE [dbo].[PyTrainScikit] (@trained_model varbinary(max)
OUTPUT)
AS
BEGIN
EXEC sp_execute_external_script
    @language = N'Python',
    @script = N'
import numpy
import pickle
from sklearn.linear_model import LogisticRegression

##Create SciKit-Learn logistic regression model
X = InputDataSet[["passenger_count", "trip_distance",
"trip_time_in_secs", "direct_distance"]]
y = numpy.ravel(InputDataSet[["tipped"]])

SKLalgo = LogisticRegression()
logitObj = SKLalgo.fit(X, y)

##Serialize model
trained_model = pickle.dumps(logitObj)
',
    @input_data_1 = N'
select tipped, fare_amount, passenger_count, trip_time_in_secs,
trip_distance,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude) as direct_distance
from nyctaxi_sample_training
',
    @input_data_1_name = N'InputDataSet',
    @params = N'@trained_model varbinary(max) OUTPUT',
    @trained_model = @trained_model OUTPUT;
;
END;
GO
```

2. Run the following SQL statements to insert the trained model into table `nyc_taxi_models`.

SQL

```
DECLARE @model VARBINARY(MAX);
EXEC PyTrainScikit @model OUTPUT;
INSERT INTO nyc_taxi_models (name, model) VALUES('SciKit_model',
@model);
```

Processing of the data and fitting the model might take a couple of minutes. Messages that would be piped to Python's **stdout** stream are displayed in the **Messages** window of Management Studio. For example:

text

```
STDOUT message(s) from external script:
C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\lib\site-
packages\revoscalepy
```

3. Open the table *nyc_taxi_models*. You can see that one new row has been added, which contains the serialized model in the column *model*.

text

```
SciKit_model
0x800363736B6C6561726E2E6C696E6561....
```

TrainTipPredictionModelRxPy

This stored procedure uses the **revoscalepy** Python package. It contains objects, transformation, and algorithms similar to those provided for the R language's **RevoScaleR** package.

By using **revoscalepy**, you can create remote compute contexts, move data between compute contexts, transform data, and train predictive models using popular algorithms such as logistic and linear regression, decision trees, and more. For more information, see [revoscalepy module in SQL Server](#) and [revoscalepy function reference](#).

1. In Management Studio, open a new **Query** window and run the following statement to create the stored procedure *TrainTipPredictionModelRxPy*. Because the stored procedure already includes a definition of the input data, you don't need to provide an input query.

SQL


```

DROP PROCEDURE IF EXISTS TrainTipPredictionModelRxPy;
GO

CREATE PROCEDURE [dbo].[TrainTipPredictionModelRxPy] (@trained_model
varbinary(max) OUTPUT)
AS
BEGIN
EXEC sp_execute_external_script
    @language = N'Python',
    @script = N'
import numpy
import pickle
from revoscalepy.functions.RxLogit import rx_logit

## Create a logistic regression model using rx_logit function from
revoscalepy package
logitObj = rx_logit("tipped ~ passenger_count + trip_distance +
trip_time_in_secs + direct_distance", data = InputDataSet);

## Serialize model
trained_model = pickle.dumps(logitObj)
',
@input_data_1 = N'
select tipped, fare_amount, passenger_count, trip_time_in_secs,
trip_distance,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude) as direct_distance
from nyctaxi_sample_training
',
@input_data_1_name = N'InputDataSet',
@params = N'@trained_model varbinary(max) OUTPUT',
@trained_model = @trained_model OUTPUT;
;
END;
GO

```

This stored procedure performs the following steps as part of model training:

- The SELECT query applies the custom scalar function *fnCalculateDistance* to calculate the direct distance between the pick-up and drop-off locations. The results of the query are stored in the default Python input variable, `InputDataset`.
- The binary variable *tipped* is used as the *label* or outcome column, and the model is fit using these feature columns: *passenger_count*, *trip_distance*, *trip_time_in_secs*, and *direct_distance*.
- The trained model is serialized and stored in the Python variable `logitObj`. By adding the T-SQL keyword OUTPUT, you can add the variable as an output of the stored procedure. In the next step, that variable is used to insert the

binary code of the model into a database table *nyc_taxi_models*. This mechanism makes it easy to store and re-use models.

2. Run the stored procedure as follows to insert the trained **revoscalepy** model into the table *nyc_taxi_models*.

SQL

```
DECLARE @model VARBINARY(MAX);
EXEC TrainTipPredictionModelRxPy @model OUTPUT;
INSERT INTO nyc_taxi_models (name, model) VALUES('revoscalepy_model',
@model);
```

Processing of the data and fitting the model might take a while. Messages that would be piped to Python's **stdout** stream are displayed in the **Messages** window of Management Studio. For example:

text

```
STDOUT message(s) from external script:
C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\lib\site-
packages\revoscalepy
```

3. Open the table *nyc_taxi_models*. You can see that one new row has been added, which contains the serialized model in the column *model*.

text

```
revoscalepy_model
0x8003637265766F7363616c....
```

In the next part of this tutorial, you'll use the trained models to create predictions.

Next steps


In this article, you:

- ✓ Created and trained a model using a SQL stored procedure
- ✓ Saved the trained model to a SQL table

Python tutorial: Run predictions using Python embedded in a stored procedure

Python tutorial: Run predictions using Python embedded in a stored procedure

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

In part five of this five-part tutorial series, you'll learn how to *operationalize* the models that you trained and saved in the previous part.

In this scenario, operationalization means deploying the model to production for scoring. The integration with SQL Server makes this fairly easy, because you can embed Python code in a stored procedure. To get predictions from the model based on new inputs, just call the stored procedure from an application and pass the new data.

This part of the tutorial demonstrates two methods for creating predictions based on a Python model: batch scoring and scoring row by row.

- **Batch scoring:** To provide multiple rows of input data, pass a SELECT query as an argument to the stored procedure. The result is a table of observations corresponding to the input cases.
- **Individual scoring:** Pass a set of individual parameter values as input. The stored procedure returns a single row or value.

All the Python code needed for scoring is provided as part of the stored procedures.

In this article, you'll:

- ✓ Create and use stored procedures for batch scoring
- ✓ Create and use stored procedures for scoring a single row

In [part one](#), you installed the prerequisites and restored the sample database.

In [part two](#), you explored the sample data and generated some plots.

In [part three](#), you learned how to create features from raw data by using a Transact-SQL function. You then called that function from a stored procedure to create a table that contains the feature values.

In [part four](#), you loaded the modules and called the necessary functions to create and train the model using a SQL Server stored procedure.

Batch scoring

The first two stored procedures created using the following scripts illustrate the basic syntax for wrapping a Python prediction call in a stored procedure. Both stored procedures require a table of data as inputs.

- The name of the model to use is provided as input parameter to the stored procedure. The stored procedure loads the serialized model from the database table `nyc_taxi_models`, using the SELECT statement in the stored procedure.
- The serialized model is stored in the Python variable `mod` for further processing using Python.
- The new cases that need to be scored are obtained from the Transact-SQL query specified in `@input_data_1`. As the query data is read, the rows are saved in the default data frame, `InputDataSet`.
- Both stored procedure use functions from `sklearn` to calculate an accuracy metric, AUC (area under curve). Accuracy metrics such as AUC can only be generated if you also provide the target label (the *tipped* column). Predictions do not need the target label (variable `y`), but the accuracy metric calculation does.

Therefore, if you don't have target labels for the data to be scored, you can modify the stored procedure to remove the AUC calculations, and return only the tip probabilities from the features (variable `x` in the stored procedure).

PredictTipSciKitPy

Run the following T-SQL statements to create the stored procedure `PredictTipSciKitPy`. This stored procedure requires a model based on the scikit-learn package, because it uses functions specific to that package.

The data frame containing inputs is passed to the `predict_proba` function of the logistic regression model, `mod`. The `predict_proba` function (`probArray = mod.predict_proba(X)`) returns a **float** that represents the probability that a tip (of any amount) will be given.

SQL

```
DROP PROCEDURE IF EXISTS PredictTipSciKitPy;
GO

CREATE PROCEDURE [dbo].[PredictTipSciKitPy] (@model varchar(50), @inquiry
nvarchar(max))
AS
BEGIN
DECLARE @lmodel2 varbinary(max) = (select model from nyc_taxi_models where
name = @model);
```

```

EXEC sp_execute_external_script
    @language = N'Python',
    @script = N'
import pickle;
import numpy;
from sklearn import metrics

mod = pickle.loads(lmodel2)
X = InputDataSet[["passenger_count", "trip_distance", "trip_time_in_secs",
"direct_distance"]]
y = numpy.ravel(InputDataSet[["tipped"]])

probArray = mod.predict_proba(X)
probList = []
for i in range(len(probArray)):
    probList.append((probArray[i])[1])

probArray = numpy.asarray(probList)
fpr, tpr, thresholds = metrics.roc_curve(y, probArray)
aucResult = metrics.auc(fpr, tpr)
print ("AUC on testing data is: " + str(aucResult))

OutputDataSet = pandas.DataFrame(data = probList, columns = ["predictions"])
',
    @input_data_1 = @inquiry,
    @input_data_1_name = N'InputDataSet',
    @params = N'@lmodel2 varbinary(max)',
    @lmodel2 = @lmodel2
WITH RESULT SETS ((Score float));
END
GO

```

PredictTipRxPy

Run the following T-SQL statements to create the stored procedure `PredictTipRxPy`.

This stored procedure uses the same inputs and creates the same type of scores as the previous stored procedure, but it uses functions from the `revoscalepy` package provided with SQL Server machine learning.

SQL

```

DROP PROCEDURE IF EXISTS PredictTipRxPy;
GO

CREATE PROCEDURE [dbo].[PredictTipRxPy] (@model varchar(50), @inquiry
nvarchar(max))
AS
BEGIN
DECLARE @lmodel2 varbinary(max) = (select model from nyc_taxi_models where
name = @model);
EXEC sp_execute_external_script

```

```

@language = N'Python',
@script = N'
import pickle;
import numpy;
from sklearn import metrics
from revoscalepy.functions.RxPredict import rx_predict;

mod = pickle.loads(lmodel2)
X = InputDataSet[["passenger_count", "trip_distance", "trip_time_in_secs",
"direct_distance"]]
y = numpy.ravel(InputDataSet[["tipped"]])

probArray = rx_predict(mod, X)
probList = probArray["tipped_Pred"].values

probArray = numpy.asarray(probList)
fpr, tpr, thresholds = metrics.roc_curve(y, probArray)
aucResult = metrics.auc(fpr, tpr)
print ("AUC on testing data is: " + str(aucResult))

OutputDataSet = pandas.DataFrame(data = probList, columns = ["predictions"])
',
@input_data_1 = @inquiry,
@input_data_1_name = N'InputDataSet',
@params = N'@lmodel2 varbinary(max)',
@lmodel2 = @lmodel2
WITH RESULT SETS ((Score float));
END
GO

```

Run batch scoring using a SELECT query

The stored procedures `PredictTipSciKitPy` and `PredictTipRxPy` require two input parameters:

- The query that retrieves the data for scoring
- The name of a trained model

By passing those arguments to the stored procedure, you can select a particular model or change the data used for scoring.

1. To use the **scikit-learn** model for scoring, call the stored procedure `PredictTipSciKitPy`, passing the model name and query string as inputs.

SQL

```

DECLARE @query_string nvarchar(max) -- Specify input query
SET @query_string='
select tipped, fare_amount, passenger_count, trip_time_in_secs,
trip_distance,

```

```
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude) as direct_distance
from nyctaxi_sample_testing'
EXEC [dbo].[PredictTipSciKitPy] 'SciKit_model', @query_string;
```

The stored procedure returns predicted probabilities for each trip that was passed in as part of the input query.

If you're using SSMS (SQL Server Management Studio) for running queries, the probabilities will appear as a table in the **Results** pane. The **Messages** pane outputs the accuracy metric (AUC or area under curve) with a value of around 0.56.

2. To use the **revoscalepy** model for scoring, call the stored procedure **PredictTipRxPy**, passing the model name and query string as inputs.

SQL

```
DECLARE @query_string nvarchar(max) -- Specify input query
SET @query_string='
select tipped, fare_amount, passenger_count, trip_time_in_secs,
trip_distance,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude) as direct_distance
from nyctaxi_sample_testing'
EXEC [dbo].[PredictTipRxPy] 'revoscalepy_model', @query_string;
```

Single-row scoring

Sometimes, instead of batch scoring, you might want to pass in a single case, getting values from an application, and returning a single result based on those values. For example, you could set up an Excel worksheet, web application, or report to call the stored procedure and pass to it inputs typed or selected by users.

In this section, you'll learn how to create single predictions by calling two stored procedures:

- **PredictTipSingleModeSciKitPy** is designed for single-row scoring using the scikit-learn model.
- **PredictTipSingleModeRxPy** is designed for single-row scoring using the revoscalepy model.
- If you haven't trained a model yet, return to [part five!](#)

Both models take as input a series of single values, such as passenger count, trip distance, and so forth. A table-valued function, `fnEngineerFeatures`, is used to convert

latitude and longitude values from the inputs to a new feature, direct distance. [Part four](#) contains a description of this table-valued function.

Both stored procedures create a score based on the Python model.

ⓘ Note

It's important that you provide all the input features required by the Python model when you call the stored procedure from an external application. To avoid errors, you might need to cast or convert the input data to a Python data type, in addition to validating data type and data length.

PredictTipSingleModeSciKitPy

The following stored procedure `PredictTipSingleModeSciKitPy` performs scoring using the `scikit-learn` model.

SQL

```
DROP PROCEDURE IF EXISTS PredictTipSingleModeSciKitPy;
GO

CREATE PROCEDURE [dbo].[PredictTipSingleModeSciKitPy] (@model varchar(50),
@passenger_count int = 0,
@trip_distance float = 0,
@trip_time_in_secs int = 0,
@pickup_latitude float = 0,
@pickup_longitude float = 0,
@dropoff_latitude float = 0,
@dropoff_longitude float = 0)
AS
BEGIN
    DECLARE @inquery nvarchar(max) = N'
SELECT * FROM [dbo].[fnEngineerFeatures](
    @passenger_count,
    @trip_distance,
    @trip_time_in_secs,
    @pickup_latitude,
    @pickup_longitude,
    @dropoff_latitude,
    @dropoff_longitude)
    '

    DECLARE @lmodel2 varbinary(max) = (select model from nyc_taxi_models where
name = @model);
EXEC sp_execute_external_script
    @language = N'Python',
    @script = N'
import pickle;
```



```

import numpy;

# Load model and unserialize
mod = pickle.loads(model)

# Get features for scoring from input data
X = InputDataSet[["passenger_count", "trip_distance", "trip_time_in_secs",
"direct_distance"]]

# Score data to get tip prediction probability as a list (of float)
probList = []
probList.append((mod.predict_proba(X)[0])[1])

# Create output data frame
OutputDataSet = pandas.DataFrame(data = probList, columns = ["predictions"])
',
    @input_data_1 = @inquiry,
    @params = N'@model varbinary(max),@passenger_count int,@trip_distance
float,
    @trip_time_in_secs int ,
    @pickup_latitude float ,
    @pickup_longitude float ,
    @dropoff_latitude float ,
    @dropoff_longitude float',
    @model = @lmodel2,
    @passenger_count =@passenger_count ,
    @trip_distance=@trip_distance,
    @trip_time_in_secs=@trip_time_in_secs,
    @pickup_latitude=@pickup_latitude,
    @pickup_longitude=@pickup_longitude,
    @dropoff_latitude=@dropoff_latitude,
    @dropoff_longitude=@dropoff_longitude
WITH RESULT SETS ((Score float));
END
GO

```

PredictTipSingleModeRxPy

The following stored procedure `PredictTipSingleModeRxPy` performs scoring using the `revoscalepy` model.

SQL

```

DROP PROCEDURE IF EXISTS PredictTipSingleModeRxPy;
GO

CREATE PROCEDURE [dbo].[PredictTipSingleModeRxPy] (@model varchar(50),
@passenger_count int = 0,
    @trip_distance float = 0,
    @trip_time_in_secs int = 0,
    @pickup_latitude float = 0,

```

```

@pickup_longitude float = 0,
@dropoff_latitude float = 0,
@dropoff_longitude float = 0)
AS
BEGIN
DECLARE @inquiry nvarchar(max) = N'
    SELECT * FROM [dbo].[fnEngineerFeatures](
        @passenger_count,
        @trip_distance,
        @trip_time_in_secs,
        @pickup_latitude,
        @pickup_longitude,
        @dropoff_latitude,
        @dropoff_longitude)
    '

DECLARE @lmodel2 varbinary(max) = (select model from nyc_taxi_models where
name = @model);
EXEC sp_execute_external_script
    @language = N'Python',
    @script = N'
import pickle;
import numpy;
from revoscalepy.functions.RxPredict import rx_predict;

# Load model and unserialize
mod = pickle.loads(model)

# Get features for scoring from input data
X = InputDataSet[["passenger_count", "trip_distance", "trip_time_in_secs",
"direct_distance"]]

# Score data to get tip prediction probability as a list (of float)

probArray = rx_predict(mod, X)

probList = []
probList = probArray["tipped_Pred"].values

# Create output data frame
OutputDataSet = pandas.DataFrame(data = probList, columns = ["predictions"])
',
    @input_data_1 = @inquiry,
    @params = N'@model varbinary(max),@passenger_count int,@trip_distance
float,
    @trip_time_in_secs int ,
    @pickup_latitude float ,
    @pickup_longitude float ,
    @dropoff_latitude float ,
    @dropoff_longitude float',
    @model = @lmodel2,
    @passenger_count =@passenger_count ,
    @trip_distance=@trip_distance,
    @trip_time_in_secs=@trip_time_in_secs,
    @pickup_latitude=@pickup_latitude,
    @pickup_longitude=@pickup_longitude,

```

```
@dropoff_latitude=@dropoff_latitude,  
@dropoff_longitude=@dropoff_longitude  
WITH RESULT SETS ((Score float));  
END  
GO
```

Generate scores from models

After the stored procedures have been created, it's easy to generate a score based on either model. Open a new **Query** window and provide parameters for each of the feature columns.

The seven required values for these feature columns are, in order:

- *passenger_count*
- *trip_distance*
- *trip_time_in_secs*
- *pickup_latitude*
- *pickup_longitude*
- *dropoff_latitude*
- *dropoff_longitude*

For example:

- To generate a prediction by using the **revoscalepy** model, run this statement:

SQL

```
EXEC [dbo].[PredictTipSingleModeRxPy] 'revoscalepy_model', 1, 2.5, 631,  
40.763958, -73.973373, 40.782139, -73.977303
```

- To generate a score by using the **scikit-learn** model, run this statement:

SQL

```
EXEC [dbo].[PredictTipSingleModeSciKitPy] 'SciKit_model', 1, 2.5, 631,  
40.763958, -73.973373, 40.782139, -73.977303
```

The output from both procedures is a probability of a tip being paid for the taxi trip with the specified parameters or features.

Conclusion

In this tutorial series, you've learned how to work with Python code embedded in stored procedures. The integration with Transact-SQL makes it much easier to deploy Python models for prediction and to incorporate model retraining as part of an enterprise data workflow.

Next steps

In this article, you:

- ✓ Created and used stored procedures for batch scoring
- ✓ Created and used stored procedures for scoring a single row

For more information about Python, see [Python extension in SQL Server](#).

Use Python with `revoscalepy` to create a model that runs remotely on SQL Server

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later

The `revoscalepy` Python library from Microsoft provides data science algorithms for data exploration, visualization, transformations, and analysis. This library has strategic importance in Python integration scenarios in SQL Server. On a multi-core server, `revoscalepy` functions can run in parallel. In a distributed architecture with a central server and client workstations (separate physical computers, all having the same `revoscalepy` library), you can write Python code that starts locally, but then shifts execution to a remote SQL Server instance where data resides.

You can find `revoscalepy` in the following Microsoft products and distributions:

- [SQL Server Machine Learning Services \(in-database\)](#)
- [Client-side Python libraries \(for development workstations\)](#)

This exercise demonstrates how to create a linear regression model based on `rx_lin_mod`, one of the algorithms in `revoscalepy` that accepts compute context as an input. The code you'll run in this exercise shifts code execution from a local to remote computing environment, enabled by `revoscalepy` functions that enable a remote compute context.

By completing this tutorial, you will learn how to:

- ✓ Use `revoscalepy` to create a linear model
- ✓ Shift operations from local to remote compute context

Prerequisites

Sample data used in this exercise is the [flightdata](#) database.

You need an IDE to run the sample code in this article, and the IDE must be linked to the Python executable.

To practice a compute context shift, you need a [local workstation](#) and a SQL Server database engine instance with [Machine Learning Services](#) and Python enabled.



If you don't have two computers, you can simulate a remote compute context on one physical computer by installing relevant applications. First, an installation of **SQL Server Machine Learning Services** operates as the "remote" instance. Second, an installation of the **Python client libraries** operates as the client. You will have two copies of the same Python distribution and Microsoft Python libraries on the same machine. You will have to keep track of file paths and which copy of the Python.exe you are using to complete the exercise successfully.

Remote compute contexts and revoscalepy

This sample demonstrates the process of creating a Python model in a remote compute context, which lets you work from a client, but choose a remote environment, such as SQL Server or Spark, where the operations are actually performed. The objective of remote compute context is to bring computation to where the data resides.

To execute Python code in SQL Server requires the **revoscalepy** package. This is a special Python package provided by Microsoft, similar to the **RevoScaleR** package for the R language. The **revoscalepy** package supports the creation of compute contexts, and provides the infrastructure for passing data and models between a local workstation and a remote server. The **revoscalepy** function that supports in-database code execution is [RxInSqlServer](#).

In this lesson, you use data in SQL Server to train a linear model based on [rx_lin_mod](#), a function in **revoscalepy** that supports regression over very large datasets.

This lesson also demonstrates the basics of how to set up and then use a **SQL Server compute context** in Python.

Run the sample code

After you have prepared the database and have the data for training stored in a table, open a Python development environment and run the code sample.

The code performs the following steps:

1. Imports the required libraries and functions.
2. Creates a connection to SQL Server. Creates **data source** objects for working with the data.
3. Modifies the data using **transformations** so that it can be used by the logistic regression algorithm.
4. Calls `rx_lin_mod` and defines the formula used to fit the model.

5. Generates a set of predictions based on the original data.
6. Creates a summary based on the predicted values.

All operations are performed using an instance of SQL Server as the compute context.

ⓘ Note

For a demonstration of this sample running from the command line, see this video: [SQL Server 2017 Advanced Analytics with Python](#)

Sample code

Python

```
from revoscalepy import RxComputeContext, RxInSqlServer, RxSqlServerData
from revoscalepy import rx_lin_mod, rx_predict, rx_summary
from revoscalepy import RxOptions, rx_import

import os

def test_linmod_sql():
    sql_server = os.getenv('PYTEST_SQL_SERVER', '.')

    sql_connection_string = 'Driver=SQL Server;Server=' + sql_server +
';Database=sqlpy;Trusted_Connection=True;'
    print("connectionString={0!s}".format(sql_connection_string))

    data_source = RxSqlServerData(
        sql_query = "select top 10 * from airlinedemosmall",
        connection_string = sql_connection_string,

        column_info = {
            "ArrDelay" : { "type" : "integer" },
            "DayOfWeek" : {
                "type" : "factor",
                "levels" : [ "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday" ]
            }
        })

    sql_compute_context = RxInSqlServer(
        connection_string = sql_connection_string,
        num_tasks = 4,
        auto_cleanup = False
    )

    #
    # Run linmod locally
    #
    linmod_local = rx_lin_mod("ArrDelay ~ DayOfWeek", data = data_source)
```

```

#
# Run linmod remotely
#
linmod = rx_lin_mod("ArrDelay ~ DayOfWeek", data = data_source,
compute_context = sql_compute_context)

# Predict results
#
predict = rx_predict(linmod, data = rx_import(input_data = data_source))
summary = rx_summary("ArrDelay ~ DayOfWeek", data = data_source,
compute_context = sql_compute_context)

```

Defining a data source vs. defining a compute context

A data source is different from a compute context. The *data source* defines the data used in your code. The compute context defines where the code will be executed. However, they use some of the same information:

- Python variables, such as `sql_query` and `sql_connection_string`, define the source of the data.

Pass these variables to the `RxSqlServerData` constructor to implement the **data source object** named `data_source`.

- You create a **compute context object** by using the `RxInSqlServer` constructor. The resulting **compute context object** is named `sql_cc`.

This example re-uses the same connection string that you used in the data source, on the assumption that the data is on the same SQL Server instance that you will be using as the compute context.

However, the data source and the compute context could be on different servers.

Changing compute contexts

After you define a compute context, you must set the **active compute context**.

By default, most operations are run locally, which means that if you don't specify a different compute context, the data will be fetched from the data source, and the code will run in your current Python environment.

There are two ways to set the active compute context:

- As an argument of a method or function
- By calling `rx_set_computecontext`

Set compute context as an argument of a method or function

In this example, you set the compute context by using an argument of the individual `rx` function.

```
linmod = rx_lin_mod_ex("ArrDelay ~ DayOfWeek", data = data, compute_context =  
sql_compute_context)
```

This compute context is reused in the call to `rxsummary`:

```
summary = rx_summary("ArrDelay ~ DayOfWeek", data = data_source, compute_context =  
sql_compute_context)
```

Set a compute context explicitly using `rx_set_compute_context`

The function `rx_set_compute_context` lets you toggle between compute contexts that have already been defined.

After you have set the active compute context, it remains active until you change it.

Using parallel processing and streaming

When you define the compute context, you can also set parameters that control how the data is handled by the compute context. These parameters differ depending on the data source type.

For SQL Server compute contexts, you can set the batch size, or provide hints about the degree of parallelism to use in running tasks.

- The sample was run on a computer with four processors, so the `num_tasks` parameter is set to 4 to allow maximum use of resources.
- If you set this value to 0, SQL Server uses the default, which is to run as many tasks in parallel as possible, under the current MAXDOP settings for the server. However, the exact number of tasks that might be allocated depends on many other factors, such as server settings, and other jobs that are running.

Next steps

These additional Python samples and tutorials demonstrate end-to-end scenarios using more complex data sources, as well as the use of remote compute contexts.

- [In-Database Python for SQL developers](#)
- [Build a predictive model using Python and SQL Server](#)

R tutorials for SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

This article describes the R tutorials and quickstarts for [SQL Server Machine Learning Services](#).

R tutorials

Tutorial	Description
Predict ski rental with decision tree	Use R and a decision tree model to predict the number of future ski rentals. Use notebooks in Azure Data Studio for preparing data and training the model, and T-SQL for model deployment.
Categorizing customers using k-means clustering	Use R to develop and deploy a K-Means clustering model to categorize customers. Use notebooks in Azure Data Studio for preparing data and training the model, and T-SQL for model deployment.
In-database R analytics for data scientists	For R developers new to SQL machine learning, this tutorial explains how to perform common data science tasks in SQL. Load and visualize data, train and save a model in a database, and use the model for predictive analytics.
In-database R analytics for SQL developers	Build and deploy a complete R solution, using only SQL tools. Focuses on moving a solution into production. You'll learn how to wrap R code in a stored procedure, save an R model in a database, and make parameterized calls to the R model for prediction.

R quickstarts

If you are new to SQL machine learning, you can also try the R quickstarts.

Quickstart	Description
Run simple R scripts	Learn the basics of how to call R in T-SQL using sp_execute_external_script .
Data structures and objects using R	Shows how SQL uses the R to handle data structures.
Create and score a predictive model in R	Explains how to create, train, and use a R model to make predictions from new data.

Next steps

- [R extension in SQL Server](#)

Tutorial: Develop a predictive model in R with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In this four-part tutorial series, you will use R and a machine learning model in [SQL Server Machine Learning Services](#) to predict the number of ski rentals.

Imagine you own a ski rental business and you want to predict the number of rentals that you'll have on a future date. This information will help you get your stock, staff, and facilities ready.

In the first part of this series, you'll get set up with the prerequisites. In parts two and three, you'll develop some R scripts in a notebook to prepare your data and train a machine learning model. Then, in part three, you'll run those R scripts inside a database using T-SQL stored procedures.

In this article, you'll learn how to:



-  Restore a sample database

In [part two](#), you'll learn how to load the data from a database into a Python data frame, and prepare the data in R.

In [part three](#), you'll learn how to train a machine learning model in R.

In [part four](#), you'll learn how to store the model in a database, and then create stored procedures from the R scripts you developed in parts two and three. The stored procedures will run on the server to make predictions based on new data.


Prerequisites

- SQL Server Machine Learning Services - To install Machine Learning Services, see the [Windows installation guide](#).
- R IDE - This tutorial uses [RStudio Desktop](#) .
- RODBC - This driver is used in the R scripts you'll develop in this tutorial. If it's not already installed, install it using the R command `install.packages("RODBC")`. For more information on RODBC, see [CRAN - Package RODBC](#) .

- SQL query tool - This tutorial assumes you're using [Azure Data Studio](#). For more information, see [How to use notebooks in Azure Data Studio](#).

Restore the sample database

The sample database used in this tutorial has been saved to a **.bak** database backup file for you to download and use.

1. Download the file [TutorialDB.bak](#) .
2. Follow the directions in [Restore a database from a backup file](#) in Azure Data Studio, using these details:
 - Import from the **TutorialDB.bak** file you downloaded
 - Name the target database "TutorialDB"
3. You can verify that the restored database exists by querying the **dbo.rental_data** table:

SQL

```
USE TutorialDB;  
SELECT * FROM [dbo].[rental_data];
```

Clean up resources

If you're not going to continue with this tutorial, delete the TutorialDB database.

Next steps

In part one of this tutorial series, you completed these steps:

- Installed the prerequisites
- Restored a sample database

To prepare the data for the machine learning model, follow part two of this tutorial series:

[Prepare data to train a predictive model in R](#)




Tutorial: Prepare data to train a predictive model in R with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In part two of this four-part tutorial series, you'll prepare data from a database using R. Later in this series, you'll use this data to train and deploy a predictive model in R with SQL Server Machine Learning Services.

In this article, you'll learn how to:

-  Restore a sample database into a database
-  Load the data from the database into an R data frame
-  Prepare the data in R by identifying some columns as categorical

In [part one](#), you learned how to restore the sample database.

In [part three](#), you'll learn how to train a machine learning model in R.

In [part four](#), you'll learn how to store the model in a database, and then create stored procedures from the R scripts you developed in parts two and three. The stored procedures will run on the server to make predictions based on new data.

Prerequisites

Part two of this tutorial assumes you have completed [part one](#) and its prerequisites.

Load the data into a data frame

To use the data in R, you'll load the data from the database into a data frame (`rentaldata`).

Create a new RScript file in RStudio and run the following script. Replace `ServerName` with your own connection information.

```
R
```

```
#Define the connection string to connect to the TutorialDB database  
connStr <- "Driver=SQL
```

```
Server;Server=ServerName;Database=TutorialDB;uid=Username;pwd=Password"
```

```
#Get the data from the table
library(RODBC)

ch <- odbcDriverConnect(connStr)

#Import the data from the table
rentaldata <- sqlFetch(ch, "dbo.rental_data")

#Take a look at the structure of the data and the top rows
head(rentaldata)
str(rentaldata)
```

You should see results similar to the following.

results

```
   Year  Month  Day  RentalCount  WeekDay  Holiday  Snow
1  2014     1   20         445         2         1     0
2  2014     2   13          40         5         0     0
3  2013     3   10         456         1         0     0
4  2014     3   31          38         2         0     0
5  2014     4   24          23         5         0     0
6  2015     2   11          42         4         0     0
'data.frame':   453 obs. of  7 variables:
 $ Year      : int  2014 2014 2013 2014 2014 2015 2013 2014 2013 2015 ...
 $ Month     : num  1 2 3 3 4 2 4 3 4 3 ...
 $ Day       : num  20 13 10 31 24 11 28 8 5 29 ...
 $ RentalCount: num  445 40 456 38 23 42 310 240 22 360 ...
 $ WeekDay   : num  2 5 1 2 5 4 1 7 6 1 ...
 $ Holiday   : int  1 0 0 0 0 0 0 0 0 0 ...
 $ Snow      : num  0 0 0 0 0 0 0 0 0 0 ...
```

Prepare the data

In this sample database, most of the preparation has already been done, but you'll do one more preparation here. Use the following R script to identify three columns as *categories* by changing the data types to *factor*.

R

```
#Changing the three factor columns to factor types
rentaldata$Holiday <- factor(rentaldata$Holiday);
rentaldata$Snow    <- factor(rentaldata$Snow);
rentaldata$WeekDay <- factor(rentaldata$WeekDay);
```



```
#Visualize the dataset after the change
str(rentaldata);
```

You should see results similar to the following.

results

```
data.frame':      453 obs. of  7 variables:
 $ Year      : int  2014 2014 2013 2014 2014 2015 2013 2014 2013 2015 ...
 $ Month     : num  1 2 3 3 4 2 4 3 4 3 ...
 $ Day       : num  20 13 10 31 24 11 28 8 5 29 ...
 $ RentalCount: num  445 40 456 38 23 42 310 240 22 360 ...
 $ WeekDay   : Factor w/ 7 levels "1","2","3","4",..: 2 5 1 2 5 4 1 7 6 1
 ...
 $ Holiday   : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 1 ...
 $ Snow      : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

The data is now prepared for training.

Clean up resources

If you're not going to continue with this tutorial, delete the TutorialDB database.

Next steps

In part two of this tutorial series, you learned how to:

- Load the sample data into an R data frame
- Prepare the data in R by identifying some columns as categorical

To create a machine learning model that uses data from the TutorialDB database, follow part three of this tutorial series:

[Create a predictive model in R with SQL machine learning](#)

Tutorial: Create a predictive model in R with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In part three of this four-part tutorial series, you'll train a predictive model in R. In the next part of this series, you'll deploy this model in a SQL Server database with Machine Learning Services.

In this article, you'll learn how to:

- ✓ Train two machine learning models
- ✓ Make predictions from both models
- ✓ Compare the results to choose the most accurate model

In [part one](#), you learned how to restore the sample database.

In [part two](#), you learned how to load the data from a database into a Python data frame and prepare the data in R.

In [part four](#), you'll learn how to store the model in a database, and then create stored procedures from the Python scripts you developed in parts two and three. The stored procedures will run in on the server to make predictions based on new data.

Prerequisites

Part three of this tutorial series assumes you have fulfilled the prerequisites of [part one](#), and completed the steps in [part two](#).

Train two models

To find the best model for the ski rental data, create two different models (linear regression and decision tree) and see which one is predicting more accurately. You'll use the data frame `rentaldata` that you created in part one of this series.

R

```
#First, split the dataset into two different sets:  
# one for training the model and the other for validating it  
train_data = rentaldata[rentaldata$Year < 2015,];  
test_data = rentaldata[rentaldata$Year == 2015,];
```

```

#Use the RentalCount column to check the quality of the prediction against
actual values
actual_counts <- test_data$RentalCount;

#Model 1: Use lm to create a linear regression model, trained with the
training data set
model_lm <- lm(RentalCount ~ Month + Day + WeekDay + Snow + Holiday, data =
train_data);

#Model 2: Use rpart to create a decision tree model, trained with the
training data set
library(rpart);
model_rpart <- rpart(RentalCount ~ Month + Day + WeekDay + Snow + Holiday,
data = train_data);

```

Make predictions from both models

Use a predict function to predict the rental counts using each trained model.

R

```

#Use both models to make predictions using the test data set.
predict_lm <- predict(model_lm, test_data)
predict_lm <- data.frame(RentalCount_Pred = predict_lm, RentalCount =
test_data$RentalCount,
                        Year = test_data$Year, Month = test_data$Month,
                        Day = test_data$Day, Weekday = test_data$WeekDay,
                        Snow = test_data$Snow, Holiday = test_data$Holiday)

predict_rpart <- predict(model_rpart, test_data)
predict_rpart <- data.frame(RentalCount_Pred = predict_rpart, RentalCount =
test_data$RentalCount,
                        Year = test_data$Year, Month = test_data$Month,
                        Day = test_data$Day, Weekday = test_data$WeekDay,
                        Snow = test_data$Snow, Holiday = test_data$Holiday)

#To verify it worked, look at the top rows of the two prediction data sets.
head(predict_lm);
head(predict_rpart);

```

results

	RentalCount_Pred	RentalCount	Month	Day	WeekDay	Snow	Holiday
1	27.45858	42	2	11	4	0	0
2	387.29344	360	3	29	1	0	0
3	16.37349	20	4	22	4	0	0
4	31.07058	42	3	6	6	0	0
5	463.97263	405	2	28	7	1	0

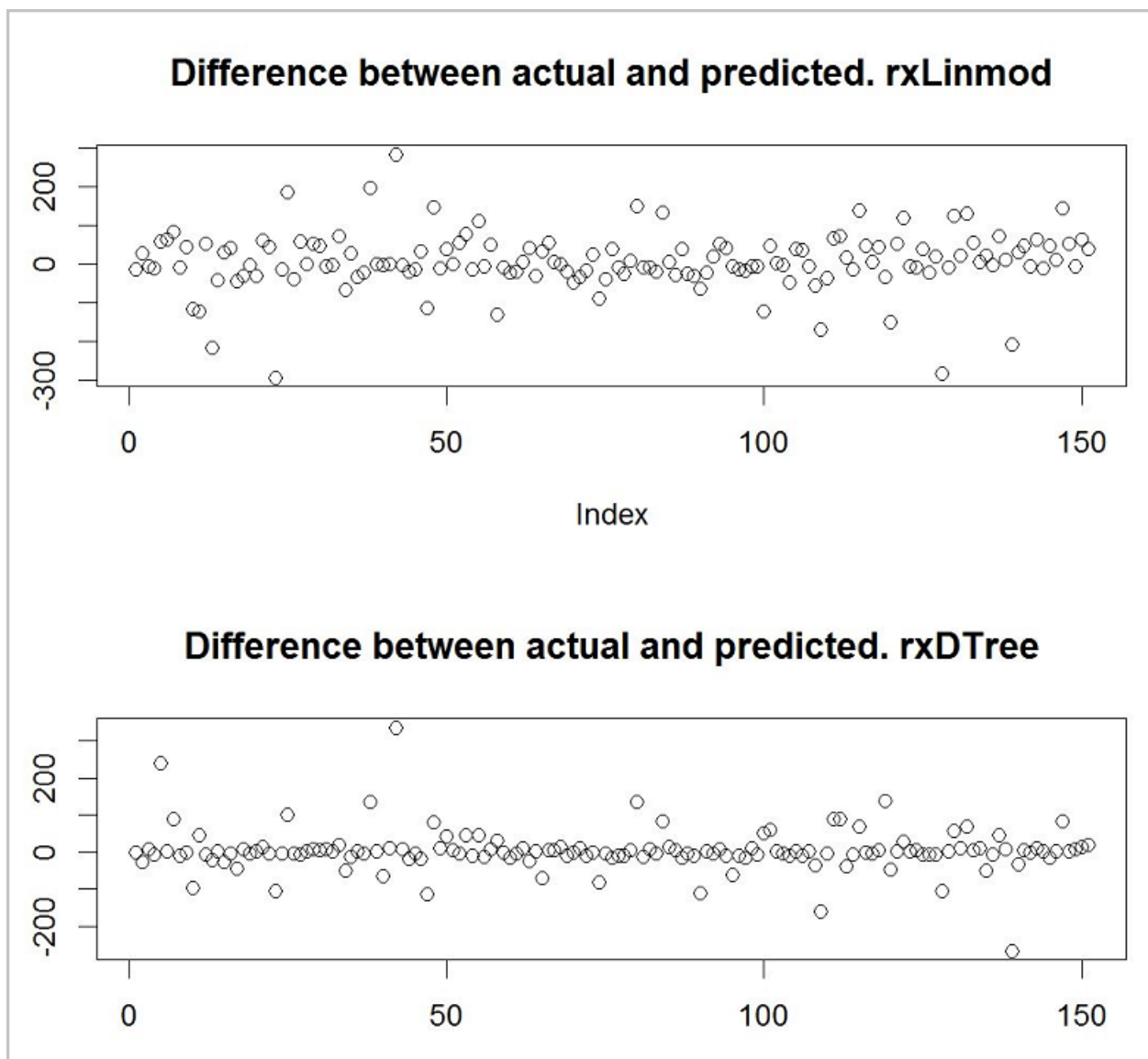
6	102.21695	38	1	12	2	1	0
	RentalCount_Pred	RentalCount	Month	Day	WeekDay	Snow	Holiday
1	40.0000	42	2	11	4	0	0
2	332.5714	360	3	29	1	0	0
3	27.7500	20	4	22	4	0	0
4	34.2500	42	3	6	6	0	0
5	645.7059	405	2	28	7	1	0
6	40.0000	38	1	12	2	1	0

Compare the results

Now you want to see which of the models gives the best predictions. A quick and easy way to do this is to use a basic plotting function to view the difference between the actual values in your training data and the predicted values.

R

```
#Use the plotting functionality in R to visualize the results from the
predictions
par(mfrow = c(1, 1));
plot(predict_lm$RentalCount_Pred - predict_lm$RentalCount, main =
"Difference between actual and predicted. lm")
plot(predict_rpart$RentalCount_Pred - predict_rpart$RentalCount, main =
"Difference between actual and predicted. rpart")
```



It looks like the decision tree model is the more accurate of the two models.

Clean up resources

If you're not going to continue with this tutorial, delete the TutorialDB database.

Next steps

In part three of this tutorial series, you learned how to:


- Train two machine learning models
- Make predictions from both models
- Compare the results to choose the most accurate model

To deploy the machine learning model you've created, follow part four of this tutorial series:

Deploy a predictive model in R with SQL machine learning





Tutorial: Deploy a predictive model in R with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In part four of this four-part tutorial series, you'll deploy a machine learning model developed in R into SQL Server using Machine Learning Services.

In this article, you'll learn how to:

-  Create a stored procedure that generates the machine learning model
-  Store the model in a database table
-  Create a stored procedure that makes predictions using the model
-  Execute the model with new data

In [part one](#), you learned how to restore the sample database.

In [part two](#), you learned how to import a sample database and then prepare the data to be used for training a predictive model in R.

In [part three](#), you learned how to create and train multiple machine learning models in R, and then choose the most accurate one.

Prerequisites

Part four of this tutorial assumes you fulfilled the prerequisites of [part one](#) and completed the steps in [part two](#) and [part three](#).

Create a stored procedure that generates the model

In part three of this tutorial series, you decided that a decision tree (dtree) model was the most accurate. Now, using the R scripts you developed, create a stored procedure (`generate_rental_model`) that trains and generates the dtree model using `rpart` from the R package.

Run the following commands in Azure Data Studio.

SQL

```

USE [TutorialDB]
DROP PROCEDURE IF EXISTS generate_rental_model;
GO
CREATE PROCEDURE generate_rental_model (@trained_model VARBINARY(max)
OUTPUT)
AS
BEGIN
    EXECUTE sp_execute_external_script @language = N'R'
        , @script = N'
rental_train_data$Month    <- factor(rental_train_data$Month);
rental_train_data$Day     <- factor(rental_train_data$Day);
rental_train_data$Holiday <- factor(rental_train_data$Holiday);
rental_train_data$Snow    <- factor(rental_train_data$Snow);
rental_train_data$WeekDay <- factor(rental_train_data$WeekDay);

#Create a dtree model and train it using the training data set
library(rpart);
model_dtree <- rpart(RentalCount ~ Month + Day + WeekDay + Snow + Holiday,
data = rental_train_data);
#Serialize the model before saving it to the database table
trained_model <- as.raw(serialize(model_dtree, connection=NULL));
'
        , @input_data_1 = N'
            SELECT RentalCount
                , Year
                , Month
                , Day
                , WeekDay
                , Snow
                , Holiday
            FROM dbo.rental_data
            WHERE Year < 2015
        '
        , @input_data_1_name = N'rental_train_data'
        , @params = N'@trained_model varbinary(max) OUTPUT'
        , @trained_model = @trained_model OUTPUT;
END;
GO

```

Store the model in a database table

Create a table in the TutorialDB database and then save the model to the table.

1. Create a table (`rental_models`) for storing the model.

SQL

```

USE TutorialDB;
DROP TABLE IF EXISTS rental_models;
GO

```



```

CREATE TABLE rental_models (
    model_name VARCHAR(30) NOT NULL DEFAULT('default model') PRIMARY
KEY
    , model VARBINARY(MAX) NOT NULL
);
GO

```

2. Save the model to the table as a binary object, with the model name "DTree".

```

SQL

-- Save model to table
TRUNCATE TABLE rental_models;

DECLARE @model VARBINARY(MAX);

EXECUTE generate_rental_model @model OUTPUT;

INSERT INTO rental_models (
    model_name
    , model
)
VALUES (
    'DTree'
    , @model
);

SELECT *
FROM rental_models;

```

Create a stored procedure that makes predictions

Create a stored procedure (`predict_rentalcount_new`) that makes predictions using the trained model and a set of new data.

```

SQL

-- Stored procedure that takes model name and new data as input parameters
and predicts the rental count for the new data
USE [TutorialDB]
DROP PROCEDURE IF EXISTS predict_rentalcount_new;
GO
CREATE PROCEDURE predict_rentalcount_new (
    @model_name VARCHAR(100)
    , @input_query NVARCHAR(MAX)
)
AS
BEGIN

```

```

DECLARE @model VARBINARY(MAX) = (
    SELECT model
    FROM rental_models
    WHERE model_name = @model_name
);

EXECUTE sp_execute_external_script @language = N'R'
    , @script = N'
#Convert types to factors
rentals$Month <- factor(rentals$Month);
rentals$Day <- factor(rentals$Day);
rentals$Holiday <- factor(rentals$Holiday);
rentals$Snow <- factor(rentals$Snow);
rentals$WeekDay <- factor(rentals$WeekDay);

#Before using the model to predict, we need to unserialize it
rental_model <- unserialize(model);

#Call prediction function
rental_predictions <- predict(rental_model, rentals);
rental_predictions <- data.frame(rental_predictions);
'

    , @input_data_1 = @input_query
    , @input_data_1_name = N'rentals'
    , @output_data_1_name = N'rental_predictions'
    , @params = N'@model varbinary(max)'
    , @model = @model
WITH RESULT SETS(("RentalCount_Predicted" FLOAT));
END;
GO

```

Execute the model with new data

Now you can use the stored procedure `predict_rentalcount_new` to predict the rental count from new data.

SQL

```

-- Use the predict_rentalcount_new stored procedure with the model name and
a set of features to predict the rental count
EXECUTE dbo.predict_rentalcount_new @model_name = 'DTree'
    , @input_query = '
    SELECT CONVERT(INT, 3) AS Month
        , CONVERT(INT, 24) AS Day
        , CONVERT(INT, 4) AS WeekDay
        , CONVERT(INT, 1) AS Snow
        , CONVERT(INT, 1) AS Holiday
    ';
GO

```

You should see a result similar to the following.

results
RentalCount_Predicted 332.571428571429

You have successfully created, trained, and deployed a model in a database. You then used that model in a stored procedure to predict values based on new data.

Clean up resources

When you've finished using the TutorialDB database, delete it from your server.

Next steps

In part four of this tutorial series, you learned how to:


- Create a stored procedure that generates the machine learning model
- Store the model in a database table
- Create a stored procedure that makes predictions using the model
- Execute the model with new data

To learn more about using R in Machine Learning Services, see:

- [Run simple R scripts](#)
- [R data structures, types and objects](#)
- [R functions](#)

Tutorial: Develop a clustering model in R with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In this four-part tutorial series, you'll use R to develop and deploy a K-Means clustering model in [SQL Server Machine Learning Services](#) to cluster customer data.

In part one of this series, you'll set up the prerequisites for the tutorial and then restore a sample dataset to a database. In parts two and three, you'll develop some R scripts in an Azure Data Studio notebook to analyze and prepare this sample data and train a machine learning model. Then, in part four, you'll run those R scripts inside a database using stored procedures.

Clustering can be explained as organizing data into groups where members of a group are similar in some way. For this tutorial series, imagine you own a retail business. You'll use the **K-Means** algorithm to perform the clustering of customers in a dataset of product purchases and returns. By clustering customers, you can focus your marketing efforts more effectively by targeting specific groups. K-Means clustering is an *unsupervised learning* algorithm that looks for patterns in data based on similarities.

In this article, you'll learn how to:


-  Restore a sample database

In [part two](#), you'll learn how to prepare the data from a database to perform clustering.

In [part three](#), you'll learn how to create and train a K-Means clustering model in R.

In [part four](#), you'll learn how to create a stored procedure in a database that can perform clustering in R based on new data.

Prerequisites

- [SQL Server Machine Learning Services](#) with the R language option - Follow the installation instructions in the [Windows installation guide](#).
- [Azure Data Studio](#). You'll use a notebook in Azure Data Studio for SQL. For more information about notebooks, see [How to use notebooks in Azure Data Studio](#).
- R IDE - This tutorial uses [RStudio Desktop](#) .

- RODBC - This driver is used in the R scripts you'll develop in this tutorial. If it's not already installed, install it using the R command `install.packages("RODBC")`. For more information on RODBC, see [CRAN - Package RODBC](#).

Restore the sample database

The sample dataset used in this tutorial has been saved to a `.bak` database backup file for you to download and use. This dataset is derived from the [tpcx-bb](#) dataset provided by the [Transaction Processing Performance Council \(TPC\)](#).

1. Download the file [tpcxbb_1gb.bak](#).
2. Follow the directions in [Restore a database from a backup file](#) in Azure Data Studio, using these details:
 - Import from the `tpcxbb_1gb.bak` file you downloaded
 - Name the target database "tpcxbb_1gb"
3. You can verify that the dataset exists after you have restored the database by querying the `dbo.customer` table:

SQL

```
USE tpcxbb_1gb;  
SELECT * FROM [dbo].[customer];
```

Clean up resources

If you're not going to continue with this tutorial, delete the `tpcxbb_1gb` database.

Next steps

In part one of this tutorial series, you completed these steps:


- Installed the prerequisites
- Restored a sample database

To prepare the data for the machine learning model, follow part two of this tutorial series:

[Prepare data to perform clustering](#)



Tutorial: Prepare data to perform clustering in R with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In part two of this four-part tutorial series, you'll prepare the data from a database to perform clustering in R with SQL Server Machine Learning Services.

In this article, you'll learn how to:

-  Separate customers along different dimensions using R
-  Load the data from the database into an R data frame

In [part one](#), you installed the prerequisites and restored the sample database.

In [part three](#), you'll learn how to create and train a K-Means clustering model in R.

In [part four](#), you'll learn how to create a stored procedure in a database that can perform clustering in R based on new data.

Prerequisites

- Part two of this tutorial assumes you have completed [part one](#).

Separate customers

Create a new RScript file in RStudio and run the following script. In the SQL query, you're separating customers along the following dimensions:

- **orderRatio** = return order ratio (total number of orders partially or fully returned versus the total number of orders)
- **itemsRatio** = return item ratio (total number of items returned versus the number of items purchased)
- **monetaryRatio** = return amount ratio (total monetary amount of items returned versus the amount purchased)
- **frequency** = return frequency

In the `connStr` function, replace `ServerName` with your own connection information.

R

```
# Define the connection string to connect to the tpcxbb_1gb database

connStr <- "Driver=SQL
Server;Server=ServerName;Database=tpcxbb_1gb;uid=Username;pwd=Password"

#Define the query to select data
input_query <- "
SELECT ss_customer_sk AS customer
      ,round(CASE
            WHEN (
                  (orders_count = 0)
                  OR (returns_count IS NULL)
                  OR (orders_count IS NULL)
                  OR ((returns_count / orders_count) IS NULL)
                )
            THEN 0.0
            ELSE (cast(returns_count AS NCHAR(10)) / orders_count)
            END, 7) AS orderRatio
      ,round(CASE
            WHEN (
                  (orders_items = 0)
                  OR (returns_items IS NULL)
                  OR (orders_items IS NULL)
                  OR ((returns_items / orders_items) IS NULL)
                )
            THEN 0.0
            ELSE (cast(returns_items AS NCHAR(10)) / orders_items)
            END, 7) AS itemsRatio
      ,round(CASE
            WHEN (
                  (orders_money = 0)
                  OR (returns_money IS NULL)
                  OR (orders_money IS NULL)
                  OR ((returns_money / orders_money) IS NULL)
                )
            THEN 0.0
            ELSE (cast(returns_money AS NCHAR(10)) / orders_money)
            END, 7) AS monetaryRatio
      ,round(CASE
            WHEN (returns_count IS NULL)
            THEN 0.0
            ELSE returns_count
            END, 0) AS frequency
FROM (
  SELECT ss_customer_sk,
         -- return order ratio
         COUNT(DISTINCT (ss_ticket_number)) AS orders_count,
         -- return ss_item_sk ratio
         COUNT(ss_item_sk) AS orders_items,
         -- return monetary amount ratio
         SUM(ss_net_paid) AS orders_money
  FROM store_sales s
```

```

        GROUP BY ss_customer_sk
      ) orders
LEFT OUTER JOIN (
  SELECT sr_customer_sk,
         -- return order ratio
         count(DISTINCT (sr_ticket_number)) AS returns_count,
         -- return ss_item_sk ratio
         COUNT(sr_item_sk) AS returns_items,
         -- return monetary amount ratio
         SUM(sr_return_amt) AS returns_money
  FROM store_returns
  GROUP BY sr_customer_sk
) returned ON ss_customer_sk = sr_customer_sk";

```

Load the data into a data frame

Now use the following script to return the results from the query to an R data frame.

```

R

# Query using input_query and get the results back
# to data frame customer_data

library(RODBC)

ch <- odbcDriverConnect(connStr)

customer_data <- sqlQuery(ch, input_query)

# Take a look at the data just loaded
head(customer_data, n = 5);

```

You should see results similar to the following.

```

results

  customer orderRatio itemsRatio monetaryRatio frequency
1    29727         0         0         0.000000         0
2    26429         0         0         0.041979         1
3    60053         0         0         0.065762         3
4    97643         0         0         0.037034         3
5    32549         0         0         0.031281         4

```

Clean up resources

If you're not going to continue with this tutorial, delete the tpcxbb_1gb database.

Next steps

In part two of this tutorial series, you learned how to:

- Separate customers along different dimensions using R
- Load the data from the database into an R data frame

To create a machine learning model that uses this customer data, follow part three of this tutorial series:

[Create a predictive model in R with SQL machine learning](#)




Tutorial: Build a clustering model in R with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In part three of this four-part tutorial series, you'll build a K-Means model in R to perform clustering. In the next part of this series, you'll deploy this model in a database with SQL Server Machine Learning Services.

In this article, you'll learn how to:

-  Define the number of clusters for a K-Means algorithm
-  Perform clustering
-  Analyze the results

In [part one](#), you installed the prerequisites and restored the sample database.

In [part two](#), you learned how to prepare the data from a database to perform clustering.

In [part four](#), you'll learn how to create a stored procedure in a database that can perform clustering in R based on new data.

Prerequisites

- Part three of this tutorial series assumes you have fulfilled the prerequisites of [part one](#) and completed the steps in [part two](#).

Define the number of clusters

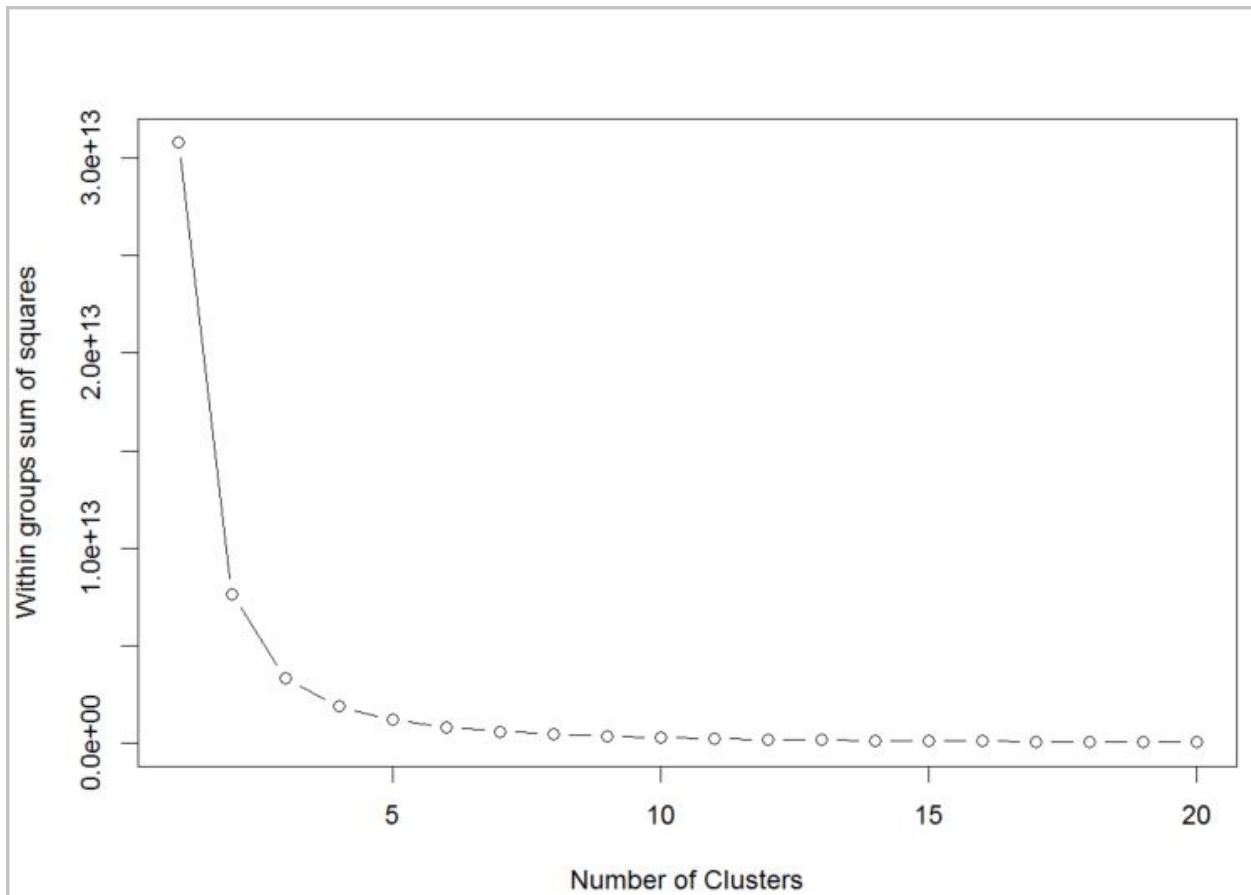
To cluster your customer data, you'll use the K-Means clustering algorithm, one of the simplest and most well-known ways of grouping data. You can read more about K-Means in [A complete guide to K-means clustering algorithm](#)^[↗].

The algorithm accepts two inputs: The data itself, and a predefined number " k " representing the number of clusters to generate. The output is k clusters with the input data partitioned among the clusters.

To determine the number of clusters for the algorithm to use, use a plot of the within groups sum of squares, by number of clusters extracted. The appropriate number of clusters to use is at the bend or "elbow" of the plot.

R

```
# Determine number of clusters by using a plot of the within groups sum of squares,  
# by number of clusters extracted.  
wss <- (nrow(customer_data) - 1) * sum(apply(customer_data, 2, var))  
for (i in 2:20)  
  wss[i] <- sum(kmeans(customer_data, centers = i)$withinss)  
plot(1:20, wss, type = "b", xlab = "Number of Clusters", ylab = "Within groups sum of squares")
```



Based on the graph, it looks like $k = 4$ would be a good value to try. That k value will group the customers into four clusters.

Perform clustering

In the following R script, you'll use the function `kmeans` to perform clustering.

R

```
# Output table to hold the customer group mappings.  
# Generate clusters using Kmeans and output key / cluster to a table  
# called return_cluster  
  
## create clustering model  
clust <- kmeans(customer_data[,2:5],4)
```

```

## create clustering output for table
customer_cluster <-
data.frame(cluster=clust$cluster,customer=customer_data$customer,orderRatio=
customer_data$orderRatio,

itemsRatio=customer_data$itemsRatio,monetaryRatio=customer_data$monetaryRati
o,frequency=customer_data$frequency)

## write cluster output to DB table
sqlSave(ch, customer_cluster, tablename = "return_cluster")

# Read the customer returns cluster table from the database
customer_cluster_check <- sqlFetch(ch, "return_cluster")

head(customer_cluster_check)

```

Analyze the results

Now that you've done the clustering using K-Means, the next step is to analyze the result and see if you can find any actionable information.

R

```

#Look at the clustering details to analyze results
clust[-1]

```

results

```

$centers
  orderRatio itemsRatio monetaryRatio frequency
1 0.621835791 0.1701519 0.35510836 1.009025
2 0.074074074 0.0000000 0.05886575 2.363248
3 0.004807692 0.0000000 0.04618708 5.050481
4 0.000000000 0.0000000 0.00000000 0.000000

```

```

$totss
[1] 40191.83

```

```

$withinss
[1] 19867.791 215.714 660.784 0.000

```

```

$tot.withinss
[1] 20744.29

```

```

$betweenss
[1] 19447.54

```

```

$size
[1] 4543 702 416 31675

```

```
$iter
[1] 3

$ifault
[1] 0
```

The four cluster means are given using the variables defined in [part two](#):

- *orderRatio* = return order ratio (total number of orders partially or fully returned versus the total number of orders)
- *itemsRatio* = return item ratio (total number of items returned versus the number of items purchased)
- *monetaryRatio* = return amount ratio (total monetary amount of items returned versus the amount purchased)
- *frequency* = return frequency

Data mining using K-Means often requires further analysis of the results, and further steps to better understand each cluster, but it can provide some good leads. Here are a couple ways you could interpret these results:

- Cluster 1 (the largest cluster) seems to be a group of customers that are not active (all values are zero).
- Cluster 3 seems to be a group that stands out in terms of return behavior.

Clean up resources

If you're not going to continue with this tutorial, delete the `tpcxbb_1gb` database.

Next steps

In part three of this tutorial series, you learned how to:

- Define the number of clusters for a K-Means algorithm
- Perform clustering
- Analyze the results

To deploy the machine learning model you've created, follow part four of this tutorial series:

[Deploy a clustering model in R with SQL machine learning](#)

Tutorial: Deploy a clustering model in R with SQL machine learning




Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In part four of this four-part tutorial series, you'll deploy a clustering model, developed in R, into a database using SQL Server Machine Learning Services.

In order to perform clustering on a regular basis, as new customers are registering, you need to be able call the R script from any app. To do that, you can deploy the R script in a database by putting the R script inside a SQL stored procedure. Because your model executes in the database, it can easily be trained against data stored in the database.

In this article, you'll learn how to:

-  Create a stored procedure that generates the model
-  Perform clustering
-  Use the clustering information

In [part one](#), you installed the prerequisites and restored the sample database.

In [part two](#), you learned how to prepare the data from a database to perform clustering.

In [part three](#), you learned how to create and train a K-Means clustering model in R.

Prerequisites

- Part four of this tutorial series assumes you have fulfilled the prerequisites of [part one](#) and completed the steps in [part two](#) and [part three](#).

Create a stored procedure that generates the model

Run the following T-SQL script to create the stored procedure. The procedure recreates the steps you developed in parts two and three of this tutorial series:

- classify customers based on their purchase and return history
- generate four clusters of customers using a K-Means algorithm

The procedure stores the resulting customer cluster mappings in the database table `customer_return_clusters`.

SQL

```
USE [tpcxbb_1gb]
DROP PROC IF EXISTS generate_customer_return_clusters;
GO
CREATE procedure [dbo].[generate_customer_return_clusters]
AS
/*
    This procedure uses R to classify customers into different groups
    based on their purchase & return history.
*/
BEGIN
    DECLARE @duration FLOAT
    , @instance_name NVARCHAR(100) = @@SERVERNAME
    , @database_name NVARCHAR(128) = db_name()
    -- Input query to generate the purchase history & return metrics
    , @input_query NVARCHAR(MAX) = N'
SELECT ss_customer_sk AS customer,
    round(CASE
        WHEN (
            (orders_count = 0)
            OR (returns_count IS NULL)
            OR (orders_count IS NULL)
            OR ((returns_count / orders_count) IS NULL)
        )
        THEN 0.0
        ELSE (cast(returns_count AS NCHAR(10)) / orders_count)
        END, 7) AS orderRatio,
    round(CASE
        WHEN (
            (orders_items = 0)
            OR (returns_items IS NULL)
            OR (orders_items IS NULL)
            OR ((returns_items / orders_items) IS NULL)
        )
        THEN 0.0
        ELSE (cast(returns_items AS NCHAR(10)) / orders_items)
        END, 7) AS itemsRatio,
    round(CASE
        WHEN (
            (orders_money = 0)
            OR (returns_money IS NULL)
            OR (orders_money IS NULL)
            OR ((returns_money / orders_money) IS NULL)
        )
        THEN 0.0
        ELSE (cast(returns_money AS NCHAR(10)) / orders_money)
        END, 7) AS monetaryRatio,
    round(CASE
        WHEN (returns_count IS NULL)
        THEN 0.0
```

```

        ELSE returns_count
        END, 0) AS frequency
FROM (
    SELECT ss_customer_sk,
           -- return order ratio
           COUNT(DISTINCT (ss_ticket_number)) AS orders_count,
           -- return ss_item_sk ratio
           COUNT(ss_item_sk) AS orders_items,
           -- return monetary amount ratio
           SUM(ss_net_paid) AS orders_money
    FROM store_sales s
    GROUP BY ss_customer_sk
    ) orders
LEFT OUTER JOIN (
    SELECT sr_customer_sk,
           -- return order ratio
           count(DISTINCT (sr_ticket_number)) AS returns_count,
           -- return ss_item_sk ratio
           COUNT(sr_item_sk) AS returns_items,
           -- return monetary amount ratio
           SUM(sr_return_amt) AS returns_money
    FROM store_returns
    GROUP BY sr_customer_sk
    ) returned ON ss_customer_sk = sr_customer_sk
,
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N'
# Define the connection string

connStr <- paste("Driver=SQL Server; Server=", instance_name,
                "; Database=", database_name,
                "; uid=Username;pwd=Password; ",
                sep="" )

# Input customer data that needs to be classified.
# This is the result we get from the query.
library(RODBC)

ch <- odbcDriverConnect(connStr);

customer_data <- sqlQuery(ch, input_query)

sqlDrop(ch, "customer_return_clusters")

## create clustering model
clust <- kmeans(customer_data[,2:5],4)

## create clustering output for table
customer_cluster <-
data.frame(cluster=clust$cluster,customer=customer_data$customer,orderRatio=
customer_data$orderRatio,

itemsRatio=customer_data$itemsRatio,monetaryRatio=customer_data$monetaryRati
o,frequency=customer_data$frequency)

```



```

## write cluster output to DB table
sqlSave(ch, customer_cluster, tablename = "customer_return_clusters")

## clean up
odbcClose(ch)
'
, @input_data_1 = N''
, @params = N'@instance_name nvarchar(100), @database_name
nvarchar(128), @input_query nvarchar(max), @duration float OUTPUT'
, @instance_name = @instance_name
, @database_name = @database_name
, @input_query = @input_query
, @duration = @duration OUTPUT;
END;

GO

```

Perform clustering

Now that you've created the stored procedure, execute the following script to perform clustering.

SQL

```

--Empty table of the results before running the stored procedure
TRUNCATE TABLE customer_return_clusters;

--Execute the clustering
--This will load the table customer_return_clusters with cluster mappings
EXECUTE [dbo].[generate_customer_return_clusters];

```

Verify that it works and that we actually have the list of customers and their cluster mappings.

SQL

```

--Select data from table customer_return_clusters
--to verify that the clustering data was loaded
SELECT TOP (5) *
FROM customer_return_clusters;

```

result

cluster	customer	orderRatio	itemsRatio	monetaryRatio	frequency
1	29727	0	0	0	0
4	26429	0	0	0.041979	1
2	60053	0	0	0.065762	3

2	97643	0	0	0.037034	3
2	32549	0	0	0.031281	4

Use the clustering information

Because you stored the clustering procedure in the database, it can perform clustering efficiently against customer data stored in the same database. You can execute the procedure whenever your customer data is updated and use the updated clustering information.

Suppose you want to send a promotional email to customers in cluster 0, the group that was inactive (you can see how the four clusters were described in [part three](#) of this tutorial). The following code selects the email addresses of customers in cluster 0.

SQL

```
USE [tpcxbb_1gb]
--Get email addresses of customers in cluster 0 for a promotion campaign
SELECT customer.[c_email_address], customer.c_customer_sk
FROM dbo.customer
JOIN
[dbo].[customer_clusters] as c
ON c.Customer = customer.c_customer_sk
WHERE c.cluster = 0
```

You can change the `c.cluster` value to return email addresses for customers in other clusters.

Clean up resources

When you're finished with this tutorial, you can delete the `tpcxbb_1gb` database.

Next steps

In part four of this tutorial series, you learned how to:

- Create a stored procedure that generates the model
- Perform clustering with SQL machine learning
- Use the clustering information


To learn more about using R in Machine Learning Services, see:

- [Run simple R scripts](#)

- R data structures, types and objects
- R functions

R tutorial: Predict NYC taxi fares with binary classification

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)



In this five-part tutorial series for SQL programmers, you'll learn about R integration in [SQL Server Machine Learning Services](#).

You'll build and deploy an R-based machine learning solution using a sample database on SQL Server. You'll use T-SQL, Azure Data Studio or SQL Server Management Studio, and a database engine instance with SQL machine learning and R language support

This tutorial series introduces you to R functions used in a data modeling workflow. Parts include data exploration, building and training a binary classification model, and model deployment. You'll use sample data from the New York City Taxi and Limousine Commission. The model you'll build predicts whether a trip is likely to result in a tip based on the time of day, distance traveled, and pick-up location.

In the first part of this series, you'll install the prerequisites and restore the sample database. In parts two and three, you'll develop some R scripts to prepare your data and train a machine learning model. Then, in parts four and five, you'll run those R scripts inside the database using T-SQL stored procedures.

In this article, you'll:


-  Install prerequisites
-  Restore the sample database

In [part two](#), you'll explore the sample data and generate some plots.

In [part three](#), you'll learn how to create features from raw data by using a Transact-SQL function. You'll then call that function from a stored procedure to create a table that contains the feature values.

In [part four](#), you'll load the modules and call the necessary functions to create and train the model using a SQL Server stored procedure.

In [part five](#), you'll learn how to operationalize the models that you trained and saved in part four.

 **Note**

This tutorial is available in both R and Python. For the Python version, see [Python tutorial: Predict NYC taxi fares with binary classification](#).

Prerequisites

- Install [SQL Server Machine Learning Services with R enabled](#)
- Install [R libraries](#)
- [Grant permissions to execute Python scripts](#)
- Restore the [NYC Taxi demo database](#)

All tasks can be done using Transact-SQL stored procedures in Azure Data Studio or Management Studio.

This tutorial assumes familiarity with basic database operations such as creating databases and tables, importing data, and writing SQL queries. It does not assume you know R and all R code is provided.

Background for SQL developers

The process of building a machine learning solution is a complex one that can involve multiple tools, and the coordination of subject matter experts across several phases:

- obtaining and cleaning data
- exploring the data and building features useful for modeling
- training and tuning the model
- deployment to production

Development and testing of the actual code is best performed using a dedicated R development environment. However, after the script is fully tested, you can easily deploy it to SQL Server using Transact-SQL stored procedures in the familiar environment of Azure Data Studio or Management Studio. Wrapping external code in stored procedures is the primary mechanism for operationalizing code in SQL Server.

After the model has been saved to the database, you can call the model for predictions from Transact-SQL by using stored procedures.

Whether you're a SQL programmer new to R, or an R developer new to SQL, this five-part tutorial series introduces a typical workflow for conducting in-database analytics with R and SQL Server.

Next steps

In this article, you:

- ✓ Installed prerequisites
- ✓ Restored the sample database

[R tutorial: Explore and visualize data](#)

R tutorial: Explore and visualize data

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In part two of this five-part tutorial series, you'll explore the sample data and generate some plots. Later, you'll learn how to serialize graphics objects in Python, and then deserialize those objects and make plots.

In part two of this five-part tutorial series, you'll review the sample data and then generate some plots using the generic `barplot` and `hist` functions in base R.

A key objective of this article is showing how to call R functions from Transact-SQL in stored procedures and save the results in application file formats:

- Create a stored procedure using `barplot` to generate an R plot as varbinary data. Use `bcp` to export the binary stream to an image file.
- Create a stored procedure using `hist` to generate a plot, saving results as JPG and PDF output.

Note

Because visualization is such a powerful tool for understanding data shape and distribution, R provides a range of functions and packages for generating histograms, scatter plots, box plots, and other data exploration graphs. R typically creates images using an R device for graphical output, which you can capture and store as a **varbinary** data type for rendering in application. You can also save the images to any of the support file formats (.JPG, .PDF, etc.).

In this article, you'll:

- ✓ Review the sample data
- ✓ Create plots using R in T-SQL
- ✓ Output plots in multiple file formats

In [part one](#), you installed the prerequisites and restored the sample database.

In [part three](#), you'll learn how to create features from raw data by using a Transact-SQL function. You'll then call that function from a stored procedure to create a table that contains the feature values.

In [part four](#), you'll load the modules and call the necessary functions to create and train the model using a SQL Server stored procedure.

In [part five](#), you'll learn how to operationalize the models that you trained and saved in part four.

Review the data

Developing a data science solution usually includes intensive data exploration and data visualization. So first take a minute to review the sample data, if you haven't already.

In the original public dataset, the taxi identifiers and trip records were provided in separate files. However, to make the sample data easier to use, the two original datasets have been joined on the columns *medallion*, *hack_license*, and *pickup_datetime*. The records were also sampled to get just 1% of the original number of records. The resulting down-sampled dataset has 1,703,957 rows and 23 columns.

Taxi identifiers

- The *medallion* column represents the taxi's unique ID number.
- The *hack_license* column contains the taxi driver's license number (anonymized).

Trip and fare records

- Each trip record includes the pickup and drop-off location and time, and the trip distance.
- Each fare record includes payment information such as the payment type, total amount of payment, and the tip amount.
- The last three columns can be used for various machine learning tasks. The *tip_amount* column contains continuous numeric values and can be used as the **label** column for regression analysis. The *tipped* column has only yes/no values and is used for binary classification. The *tip_class* column has multiple **class labels** and therefore can be used as the label for multi-class classification tasks.

This walkthrough demonstrates only the binary classification task; you are welcome to try building models for the other two machine learning tasks, regression and multiclass classification.

- The values used for the label columns are all based on the *tip_amount* column, using these business rules:
-

Derived column name	Rule
tipped	If tip_amount > 0, tipped = 1, otherwise tipped = 0
tip_class	Class 0: tip_amount = \$0 Class 1: tip_amount > \$0 and tip_amount <= \$5 Class 2: tip_amount > \$5 and tip_amount <= \$10 Class 3: tip_amount > \$10 and tip_amount <= \$20 Class 4: tip_amount > \$20

Create plots using R in T-SQL

To create the plot, use the R function `barplot`. This step plots a histogram based on data from a Transact-SQL query. You can wrap this function in a stored procedure, `RPlotHistogram`.

1. In SQL Server Management Studio, in Object Explorer, right-click the `NYCTaxi_Sample` database and select **New Query**. Or, in Azure Data Studio, select **New Notebook** from the **File** menu and connect to the database.
2. Paste in the following script to create a stored procedure that plots the histogram. This example is named `RPlotHistogram`.

SQL

```
CREATE PROCEDURE [dbo].[RPlotHistogram]
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @query nvarchar(max) =
    N'SELECT tipped FROM [dbo].[nyctaxi_sample]'
    EXECUTE sp_execute_external_script @language = N'R',
    @script = N'

    image_file = tempfile();
    jpeg(filename = image_file);
    #Plot histogram
    barplot(table(InputDataSet$tipped), main = "Tip Histogram",
    col="lightgreen", xlab="Tipped or not", ylab = "Counts", space=0)
    dev.off();
    OutputDataSet <- data.frame(data=readBin(file(image_file, "rb"),
    what=raw(), n=1e6));
    ',
    @input_data_1 = @query
    WITH RESULT SETS ((plot varbinary(max)));
```

```
END
GO
```

Key points to understand in this script include the following:

- The variable `@query` defines the query text (`'SELECT tipped FROM nyctaxi_sample'`), which is passed to the R script as the argument to the script input variable, `@input_data_1`. For R scripts that run as external processes, you should have a one-to-one mapping between inputs to your script, and inputs to the `sp_execute_external_script` system stored procedure that starts the R session on SQL Server.
- Within the R script, a variable (`image_file`) is defined to store the image.
- The `barplot` function is called to generate the plot.
- The R device is set to **off** because you are running this command as an external script in SQL Server. Typically in R, when you issue a high-level plotting command, R opens a graphics window, called a *device*. You can turn the device off if you are writing to a file or handling the output some other way.
- The R graphics object is serialized to an R data.frame for output.

Execute the stored procedure and use bcp to export binary data to an image file

The stored procedure returns the image as a stream of varbinary data, which obviously you cannot view directly. However, you can use the **bcp** utility to get the varbinary data and save it as an image file on a client computer.

1. In Management Studio, run the following statement:

```
SQL
EXEC [dbo].[RPlotHistogram]
```

Results

```
plot 0xFFD8FFE000104A4649...
```

2. Open a PowerShell command prompt and run the following command, providing the appropriate instance name, database name, username, and credentials as arguments. For those using Windows identities, you can replace **-U** and **-P** with **-T**.

PowerShell

```
bcp "exec RPlotHistogram" queryout "plot.jpg" -S <SQL Server instance name> -d NYCTaxi_Sample -U <user name> -P <password> -T
```

ⓘ Note

Command switches for bcp are case-sensitive.

3. If the connection is successful, you will be prompted to enter more information about the graphic file format.

Press ENTER at each prompt to accept the defaults, except for these changes:

- For **prefix-length of field plot**, type 0.
- Type **Y** if you want to save the output parameters for later reuse.

text

```
Enter the file storage type of field plot [varbinary(max)]:
Enter prefix-length of field plot [8]: 0
Enter length of field plot [0]:
Enter field terminator [none]:

Do you want to save this format information in a file? [Y/n]
Host filename [bcp.fmt]:
```

Results

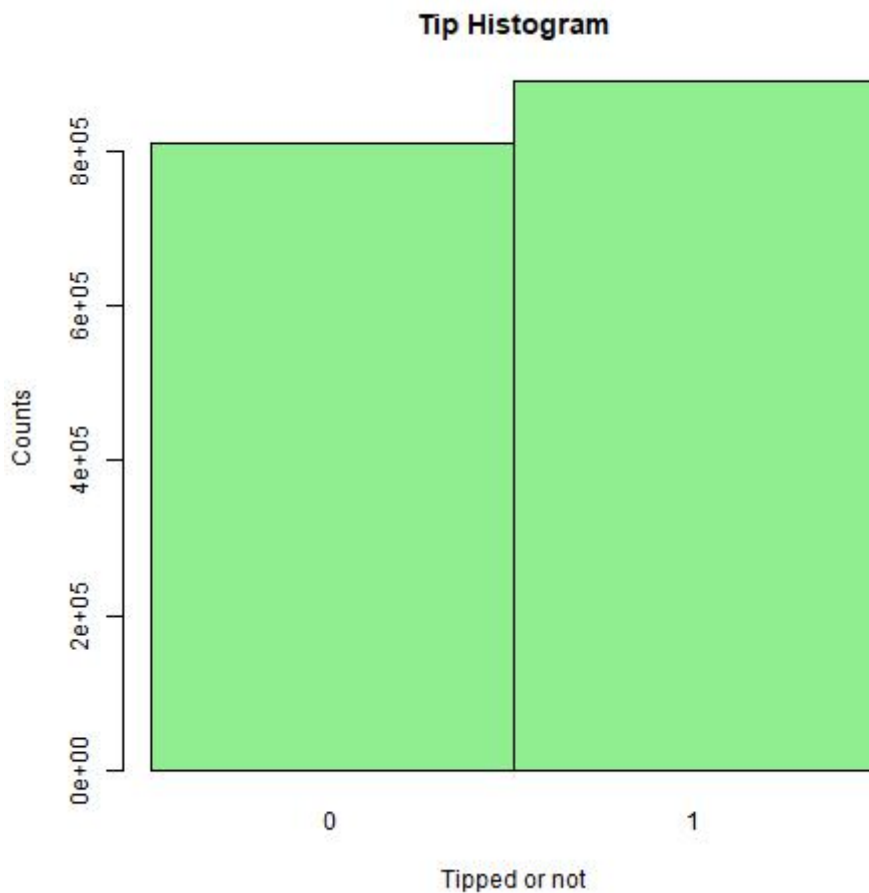
text

```
Starting copy...
1 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total      : 3922   Average : (0.25 rows per sec.)
```

💡 Tip

If you save the format information to file (bcp.fmt), the **bcp** utility generates a format definition that you can apply to similar commands in future without being prompted for graphic file format options. To use the format file, add `-f bcp.fmt` to the end of any command line, after the password argument.

- The output file will be created in the same directory where you ran the PowerShell command. To view the plot, just open the file plot.jpg.



Create a stored procedure using `hist`

Typically, data scientists generate multiple data visualizations to get insights into the data from different perspectives. In this example, you will create a stored procedure called `RPlotHist` to write histograms, scatterplots, and other R graphics to .JPG and .PDF format.

This stored procedure uses the `hist` function to create the histogram, exporting the binary data to popular formats such as .JPG, .PDF, and .PNG.

- In SQL Server Management Studio, in Object Explorer, right-click the `NYCTaxi_Sample` database and select **New Query**.
- Paste in the following script to create a stored procedure that plots the histogram. This example is named `RPlotHist`.

SQL

```
CREATE PROCEDURE [dbo].[RPlotHist]
AS
```

```

BEGIN
  SET NOCOUNT ON;
  DECLARE @query nvarchar(max) =
  N'SELECT cast(tipped as int) as tipped, tip_amount, fare_amount FROM
[dbo].[nyctaxi_sample]'
  EXECUTE sp_execute_external_script @language = N'R',
  @script = N'
    # Set output directory for files and check for existing files with
same names
    mainDir <- 'C:\\temp\\plots'
    dir.create(mainDir, recursive = TRUE, showWarnings = FALSE)
    setwd(mainDir);
    print("Creating output plot files:", quote=FALSE)

    # Open a jpeg file and output histogram of tipped variable in that
file.
    dest_filename = tempfile(pattern = 'rHistogram_Tipped_', tmpdir =
mainDir)
    dest_filename = paste(dest_filename, '.jpg', sep="")
    print(dest_filename, quote=FALSE);
    jpeg(filename=dest_filename);
    hist(InputDataSet$tipped, col = 'lightgreen', xlab='Tipped',
        ylab = 'Counts', main = 'Histogram, Tipped');
    dev.off();

    # Open a pdf file and output histograms of tip amount and fare
amount.
    # Outputs two plots in one row
    dest_filename = tempfile(pattern =
'rHistograms_Tip_and_Fare_Amount_', tmpdir = mainDir)
    dest_filename = paste(dest_filename, '.pdf', sep="")
    print(dest_filename, quote=FALSE);
    pdf(file=dest_filename, height=4, width=7);
    par(mfrow=c(1,2));
    hist(InputDataSet$tip_amount, col = 'lightgreen',
        xlab='Tip amount ($)',
        ylab = 'Counts',
        main = 'Histogram, Tip amount', xlim = c(0,40), 100);
    hist(InputDataSet$fare_amount, col = 'lightgreen',
        xlab='Fare amount ($)',
        ylab = 'Counts',
        main = 'Histogram,
        Fare amount',
        xlim = c(0,100), 100);
    dev.off();

    # Open a pdf file and output an xyplot of tip amount vs. fare
amount using lattice;
    # Only 10,000 sampled observations are plotted here, otherwise file
is large.
    dest_filename = tempfile(pattern =
'rXYPlots_Tip_vs_Fare_Amount_', tmpdir = mainDir)
    dest_filename = paste(dest_filename, '.pdf', sep="")
    print(dest_filename, quote=FALSE);
    pdf(file=dest_filename, height=4, width=4);

```

```

plot(tip_amount ~ fare_amount,
     data = InputDataSet[sample(nrow(InputDataSet), 10000), ],
     ylim = c(0,50),
     xlim = c(0,150),
     cex=.5,
     pch=19,
     col='darkgreen',
     main = 'Tip amount by Fare amount',
     xlab='Fare Amount ($)',
     ylab = 'Tip Amount ($)');
dev.off();',
@input_data_1 = @query
END

```

Key points to understand in this script include the following:

- The output of the SELECT query within the stored procedure is stored in the default R data frame, `InputDataSet`. Various R plotting functions can then be called to generate the actual graphics files. Most of the embedded R script represents options for these graphics functions, such as `plot` or `hist`.
- The R device is set to **off** because you are running this command as an external script in SQL Server. Typically in R, when you issue a high-level plotting command, R opens a graphics window, called a *device*. You can turn the device off if you are writing to a file or handling the output some other way.
- All files are saved to the local folder `C:\temp\Plots`. The destination folder is defined by the arguments provided to the R script as part of the stored procedure. To output the files to a different folder, change the value of the `mainDir` variable in the R script embedded in the stored procedure. You can also modify the script to output different formats, more files, and so on.

Execute the stored procedure

Run the following statement to export binary plot data to JPEG and PDF file formats.

```
SQL
```

```
EXEC RPlotHist
```

Results

```
text
```

```

STDOUT message(s) from external script:
[1] Creating output plot files:[1]

```

```
C:\temp\plots\rHistogram_Tipped_18887f6265d4.jpg[1]
```

```
C:\temp\plots\rHistograms_Tip_and_Fare_Amount_1888441e542c.pdf[1]
```

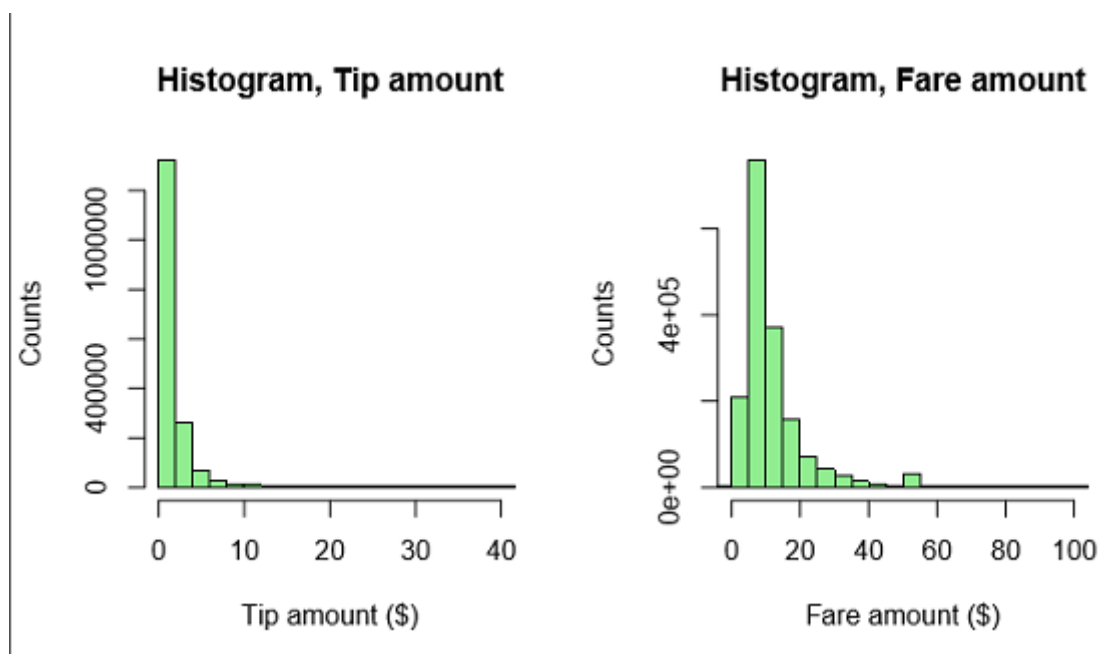
```
C:\temp\plots\rXYPlots_Tip_vs_Fare_Amount_18887c9d517b.pdf
```

The numbers in the file names are randomly generated to ensure that you don't get an error when trying to write to an existing file.

View output

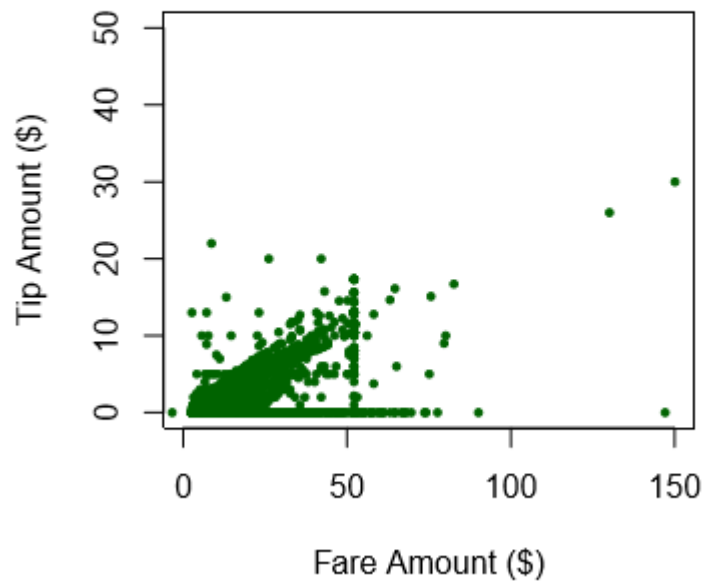
To view the plot, open the destination folder and review the files that were created by the R code in the stored procedure.

1. Go the folder indicated in the STDOUT message (in the example, this is C:\temp\plots)
2. Open `rHistogram_Tipped.jpg` to show the number of trips that got a tip vs. the trips that got no tip (this histogram is similar to the one you generated in the previous step).
3. Open `rHistograms_Tip_and_Fare_Amount.pdf` to view distribution of tip amounts, plotted against the fare amounts.



4. Open `rXYPlots_Tip_vs_Fare_Amount.pdf` to view a scatterplot with the fare amount on the x-axis and the tip amount on the y-axis.

Tip amount by Fare amount



Next steps

In this article, you:

- ✓ Reviewed the sample data
- ✓ Created plots using R in T-SQL
- ✓ Output plots in multiple file formats

[R tutorial: Create data features](#)



R tutorial: Create data features

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In part three of this five-part tutorial series, you'll learn how to create features from raw data by using a Transact-SQL function. You'll then call that function from a SQL stored procedure to create a table that contains the feature values.

In this article, you'll:

-  Modify a custom function to calculate trip distance
-  Save the features using another custom function

In [part one](#), you installed the prerequisites and restored the sample database.

In [part two](#), you reviewed the sample data and generated some plots.


In [part four](#), you'll load the modules and call the necessary functions to create and train the model using a SQL Server stored procedure.

In [part five](#), you'll learn how to operationalize the models that you trained and saved in part four.

In [part five](#), you'll learn how to operationalize the models that you trained and saved in part four.

About feature engineering

After several rounds of data exploration, you have collected some insights from the data, and are ready to move on to *feature engineering*. This process of creating meaningful features from the raw data is a critical step in creating analytical models.

In this dataset, the distance values are based on the reported meter distance, and don't necessarily represent geographical distance or the actual distance traveled. Therefore, you'll need to calculate the direct distance between the pick-up and drop-off points, by using the coordinates available in the source NYC Taxi dataset. You can do this by using the [Haversine formula](#)  in a custom Transact-SQL function.

You'll use one custom T-SQL function, *fnCalculateDistance*, to compute the distance using the Haversine formula, and use a second custom T-SQL function, *fnEngineerFeatures*, to create a table containing all the features.

The overall process is as follows:

- Create the T-SQL function that performs the calculations
- Call the function to generate the feature data
- Save the feature data to a table

Calculate trip distance using `fnCalculateDistance`

The function `fnCalculateDistance` should have been downloaded and registered with SQL Server as part of the preparation for this tutorial. Take a minute to review the code.

1. In Management Studio, expand **Programmability**, expand **Functions** and then **Scalar-valued functions**.
2. Right-click `fnCalculateDistance`, and select **Modify** to open the Transact-SQL script in a new query window.

SQL

```
CREATE FUNCTION [dbo].[fnCalculateDistance] (@Lat1 float, @Long1 float,
@Lat2 float, @Long2 float)
-- User-defined function that calculates the direct distance between
two geographical coordinates.
RETURNS float
AS
BEGIN
    DECLARE @distance decimal(28, 10)
    -- Convert to radians
    SET @Lat1 = @Lat1 / 57.2958
    SET @Long1 = @Long1 / 57.2958
    SET @Lat2 = @Lat2 / 57.2958
    SET @Long2 = @Long2 / 57.2958
    -- Calculate distance
    SET @distance = (SIN(@Lat1) * SIN(@Lat2)) + (COS(@Lat1) * COS(@Lat2)
* COS(@Long2 - @Long1))
    --Convert to miles
    IF @distance <> 0
    BEGIN
        SET @distance = 3958.75 * ATAN(SQRT(1 - POWER(@distance, 2)) /
@distance);
    END
    RETURN @distance
END
GO
```

- The function is a scalar-valued function, returning a single data value of a predefined type.
- It takes latitude and longitude values as inputs, obtained from trip pick-up and drop-off locations. The Haversine formula converts locations to radians and uses those values to compute the direct distance in miles between those two locations.

Generate the features using *fnEngineerFeatures*

To add the computed values to a table that can be used for training the model, you'll use another function, *fnEngineerFeatures*. The new function calls the previously created T-SQL function, *fnCalculateDistance*, to get the direct distance between pick-up and drop-off locations.

1. Take a minute to review the code for the custom T-SQL function, *fnEngineerFeatures*, which should have been created for you as part of the preparation for this walkthrough.

SQL

```
CREATE FUNCTION [dbo].[fnEngineerFeatures] (
  @passenger_count int = 0,
  @trip_distance float = 0,
  @trip_time_in_secs int = 0,
  @pickup_latitude float = 0,
  @pickup_longitude float = 0,
  @dropoff_latitude float = 0,
  @dropoff_longitude float = 0)
RETURNS TABLE
AS
RETURN
(
  -- Add the SELECT statement with parameter references here
  SELECT
    @passenger_count AS passenger_count,
    @trip_distance AS trip_distance,
    @trip_time_in_secs AS trip_time_in_secs,
    [dbo].[fnCalculateDistance](@pickup_latitude, @pickup_longitude,
    @dropoff_latitude, @dropoff_longitude) AS direct_distance
)
GO
```

- This table-valued function that takes multiple columns as inputs, and outputs a table with multiple feature columns.

- The purpose of this function is to create new features for use in building a model.

2. To verify that this function works, use it to calculate the geographical distance for those trips where the metered distance was 0 but the pick-up and drop-off locations were different.

SQL

```
SELECT tipped, fare_amount, passenger_count, (trip_time_in_secs/60)
as TripMinutes,
    trip_distance, pickup_datetime, dropoff_datetime,
    dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude) AS direct_distance
FROM nyctaxi_sample
WHERE pickup_longitude != dropoff_longitude and pickup_latitude !=
dropoff_latitude and trip_distance = 0
ORDER BY trip_time_in_secs DESC
```

As you can see, the distance reported by the meter doesn't always correspond to geographical distance. This is why feature engineering is so important. You can use these improved data features to train a machine learning model using R.

Next steps

In this article, you:

- ✓ Modified a custom function to calculate trip distance
- ✓ Saved the features using another custom function

[R tutorial: Train and save model](#)



R tutorial: Train and save model

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In part four of this five-part tutorial series, you'll learn how to train a machine learning model by using R. You'll train the model using the data features you created in the previous part, and then save the trained model in a SQL Server table. In this case, the R packages are already installed with R Services (In-Database), so everything can be done from SQL.

In this article, you'll:

-  Create and train a model using a SQL stored procedure
-  Save the trained model to a SQL table

In [part one](#), you installed the prerequisites and restored the sample database.

In [part two](#), you reviewed the sample data and generate some plots.

In [part three](#), you learned how to create features from raw data by using a Transact-SQL function. You then called that function from a stored procedure to create a table that contains the feature values.

In [part five](#), you'll learn how to operationalize the models that you trained and saved in part four.

Create the stored procedure

When calling R from T-SQL, you use the system stored procedure, [sp_execute_external_script](#). However, for processes that you repeat often, such as retraining a model, it is easier to encapsulate the call to `sp_execute_external_script` in another stored procedure.

1. In Management Studio, open a new **Query** window.
2. Run the following statement to create the stored procedure **RTrainLogitModel**. This stored procedure defines the input data and uses **glm** to create a logistic regression model.

```
SQL
```

```

CREATE PROCEDURE [dbo].[RTrainLogitModel] (@trained_model
varbinary(max) OUTPUT)

AS
BEGIN
    DECLARE @inquiry nvarchar(max) = N'
        select tipped, fare_amount,
passenger_count,trip_time_in_secs,trip_distance,
        pickup_datetime, dropoff_datetime,
        dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude) as direct_distance
        from nyctaxi_sample
        tablesample (70 percent) repeatable (98052)
    '

    EXEC sp_execute_external_script @language = N'R',
        @script = N'

## Create model
logitObj <- glm(tipped ~ passenger_count + trip_distance +
trip_time_in_secs + direct_distance, data = InputDataSet, family =
binomial)
summary(logitObj)

## Serialize model
trained_model <- as.raw(serialize(logitObj, NULL));
',
        @input_data_1 = @inquiry,
        @params = N'@trained_model varbinary(max) OUTPUT',
        @trained_model = @trained_model OUTPUT;
END
GO

```

- To ensure that some data is left over to test the model, 70% of the data are randomly selected from the taxi data table for training purposes.
- The SELECT query uses the custom scalar function *fnCalculateDistance* to calculate the direct distance between the pick-up and drop-off locations. The results of the query are stored in the default R input variable, `InputDataset`.
- The R script calls the R function `glm` to create the logistic regression model.

The binary variable *tipped* is used as the *label* or outcome column, and the model is fit using these feature columns: *passenger_count*, *trip_distance*, *trip_time_in_secs*, and *direct_distance*.

- The trained model, saved in the R variable `logitObj`, is serialized and returned as an output parameter.

Train and deploy the R model using the stored procedure

Because the stored procedure already includes a definition of the input data, you don't need to provide an input query.

1. To train and deploy the R model, call the stored procedure and insert it into the database table *nyc_taxi_models*, so that you can use it for future predictions:

SQL

```
DECLARE @model VARBINARY(MAX);
EXEC RTrainLogitModel @model OUTPUT;
INSERT INTO nyc_taxi_models (name, model) VALUES('RTrainLogit_model',
@model);
```

2. Watch the **Messages** window of Management Studio for messages that would be piped to R's **stdout** stream, like this message:

```
"STDOUT message(s) from external script: Rows Read: 1193025, Total Rows
Processed: 1193025, Total Chunk Time: 0.093 seconds"
```

3. When the statement has completed, open the table *nyc_taxi_models*. Processing of the data and fitting the model might take a while.

You can see that one new row has been added, which contains the serialized model in the column *model* and the model name **RTrainLogit_model** in the column *name*.

text

model	name
0x580A0000002000302020....	RTrainLogit_model

In the next part of this tutorial you'll use the trained model to generate predictions.

Next steps

In this article, you:

- ✓ Created and trained a model using a SQL stored procedure
- ✓ Saved the trained model to a SQL table

R tutorial: Run predictions in SQL stored procedures

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In part five of this five-part tutorial series, you'll learn to *operationalize* the model that you trained and saved in the previous part by using the model to predict potential outcomes. The model is wrapped in a stored procedure which can be called directly by other applications.

This article demonstrates two ways to perform scoring:

- **Batch scoring mode:** Use a SELECT query as an input to the stored procedure. The stored procedure returns a table of observations corresponding to the input cases.
- **Individual scoring mode:** Pass a set of individual parameter values as input. The stored procedure returns a single row or value.

In this article, you'll:

- ✓ Create and use stored procedures for batch scoring
- ✓ Create and use stored procedures for scoring a single row

In [part one](#), you installed the prerequisites and restored the sample database.

In [part two](#), you reviewed the sample data and generated some plots.

In [part three](#), you learned how to create features from raw data by using a Transact-SQL function. You then called that function from a stored procedure to create a table that contains the feature values.

In [part four](#), you loaded the modules and called the necessary functions to create and train the model using a SQL Server stored procedure.

Basic scoring

The stored procedure `RPredict` illustrates the basic syntax for wrapping a `PREDICT` call in a stored procedure.

```

CREATE PROCEDURE [dbo].[Rpredict] (@model varchar(250), @inquiry
nvarchar(max))
AS
BEGIN

DECLARE @lmodel2 varbinary(max) = (SELECT model FROM nyc_taxi_models WHERE
name = @model);
EXEC sp_execute_external_script @language = N'R',
    @script = N'
        mod <- unserialize(as.raw(model));
        print(summary(mod))
        OutputDataSet <- data.frame(predict(mod, InputDataSet, type =
"response"));
        str(OutputDataSet)
        print(OutputDataSet)
    ',
    @input_data_1 = @inquiry,
    @params = N'@model varbinary(max)',
    @model = @lmodel2
    WITH RESULT SETS (("Score" float));
END
GO

```

- The SELECT statement gets the serialized model from the database, and stores the model in the R variable `mod` for further processing using R.
- The new cases for scoring are obtained from the Transact-SQL query specified in `@inquiry`, the first parameter to the stored procedure. As the query data is read, the rows are saved in the default data frame, `InputDataSet`. This data frame is passed to the `PREDICT` function which generates the scores.

```
OutputDataSet <- data.frame(predict(mod, InputDataSet, type = "response"));
```

Because a data.frame can contain a single row, you can use the same code for batch or single scoring.

- The value returned by the `PREDICT` function is a **float** that represents the probability that the driver gets a tip of any amount.

Batch scoring (a list of predictions)

A more common scenario is to generate predictions for multiple observations in batch mode. In this step, let's see how batch scoring works.

1. Start by getting a smaller set of input data to work with. This query creates a "top 10" list of trips with passenger count and other features needed to make a

prediction.

SQL

```
SELECT TOP 10 a.passenger_count AS passenger_count, a.trip_time_in_secs
AS trip_time_in_secs, a.trip_distance AS trip_distance,
a.dropoff_datetime AS dropoff_datetime,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude,dropoff_longitude) AS direct_distance

FROM (SELECT medallion, hack_license, pickup_datetime,
passenger_count,trip_time_in_secs,trip_distance, dropoff_datetime,
pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
FROM nyctaxi_sample)a

LEFT OUTER JOIN

(SELECT medallion, hack_license, pickup_datetime FROM nyctaxi_sample
TABLESAMPLE (70 percent) REPEATABLE (98052) )b

ON a.medallion=b.medallion AND a.hack_license=b.hack_license
AND a.pickedup_datetime=b.pickedup_datetime
WHERE b.medallion IS NULL
```

Sample results

text

passenger_count	trip_time_in_secs	trip_distance	dropoff_datetime
1	283	0.7	2013-03-27
14:54:50.000	0.5427964547		
1	289	0.7	2013-02-24
12:55:29.000	0.3797099614		
1	214	0.7	2013-06-26
13:28:10.000	0.6970098661		

2. Create a stored procedure called **RPredictBatchOutput** in Management Studio.

SQL

```
CREATE PROCEDURE [dbo].[RPredictBatchOutput] (@model varchar(250),
@inquiry nvarchar(max))
AS
BEGIN
DECLARE @lmodel2 varbinary(max) = (SELECT model FROM nyc_taxi_models
WHERE name = @model);
EXEC sp_execute_external_script
@language = N'R',
@script = N'
mod <- unserialize(as.raw(model));
```

```

print(summary(mod))
OutputDataSet <- data.frame(predict(mod, InputDataSet, type =
"response"));
str(OutputDataSet)
print(OutputDataSet)
',
@input_data_1 = @inquiry,
@params = N'@model varbinary(max)',
@model = @lmodel2
WITH RESULT SETS ((Score float));
END

```

3. Provide the query text in a variable and pass it as a parameter to the stored procedure:

```

SQL

-- Define the input data
DECLARE @query_string nvarchar(max)
SET @query_string='SELECT TOP 10 a.passenger_count as passenger_count,
a.trip_time_in_secs AS trip_time_in_secs, a.trip_distance AS
trip_distance, a.dropoff_datetime AS dropoff_datetime,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude,dropoff_longitude) AS direct_distance FROM (SELECT
medallion, hack_license, pickup_datetime,
passenger_count,trip_time_in_secs,trip_distance, dropoff_datetime,
pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
FROM nyctaxi_sample )a LEFT OUTER JOIN (SELECT medallion,
hack_license, pickup_datetime FROM nyctaxi_sample TABLESAMPLE (70
percent) REPEATABLE (98052))b ON a.medallion=b.medallion AND
a.hack_license=b.hack_license AND a.pickup_datetime=b.pickup_datetime
WHERE b.medallion is null'

-- Call the stored procedure for scoring and pass the input data
EXEC [dbo].[RPredictBatchOutput] @model = 'RTrainLogit_model', @inquiry
= @query_string;

```

The stored procedure returns a series of values representing the prediction for each of the top 10 trips. However, the top trips are also single-passenger trips with a relatively short trip distance, for which the driver is unlikely to get a tip.

Tip

Rather than returning just the "yes-tip" and "no-tip" results, you could also return the probability score for the prediction, and then apply a WHERE clause to the Score column values to categorize the score as "likely to tip" or "unlikely to tip", using a threshold value such as 0.5 or 0.7. This step is not included in the stored procedure but it would be easy to implement.

Single-row scoring of multiple inputs

Sometimes you want to pass in multiple input values and get a single prediction based on those values. For example, you could set up an Excel worksheet, web application, or Reporting Services report to call the stored procedure and provide inputs typed or selected by users from those applications.

In this section, you learn how to create single predictions using a stored procedure that takes multiple inputs, such as passenger count, trip distance, and so forth. The stored procedure creates a score based on the previously stored R model.

If you call the stored procedure from an external application, make sure that the data matches the requirements of the R model. This might include ensuring that the input data can be cast or converted to an R data type, or validating data type and data length.

1. Create a stored procedure `RPredictSingleRow`.

SQL

```
CREATE PROCEDURE [dbo].[RPredictSingleRow] @model varchar(50),
@passenger_count int = 0, @trip_distance float = 0, @trip_time_in_secs
int = 0, @pickup_latitude float = 0, @pickup_longitude float = 0,
@dropoff_latitude float = 0, @dropoff_longitude float = 0
AS
BEGIN
DECLARE @inquiry nvarchar(max) = N'SELECT * FROM [dbo].
[fnEngineerFeatures](@passenger_count, @trip_distance,
@trip_time_in_secs, @pickup_latitude, @pickup_longitude,
@dropoff_latitude, @dropoff_longitude)';
DECLARE @lmodel2 varbinary(max) = (SELECT model FROM nyc_taxi_models
WHERE name = @model);
EXEC sp_execute_external_script
    @language = N'R',
    @script = N'
        mod <- unserialize(as.raw(model));
        print(summary(mod));
        OutputDataSet <- data.frame(predict(mod, InputDataSet, type =
"response"));
        str(OutputDataSet);
        print(OutputDataSet);
    ',
    @input_data_1 = @inquiry,
    @params = N'@model varbinary(max),@passenger_count int,@trip_distance
float,@trip_time_in_secs int , @pickup_latitude float
,@pickup_longitude float ,@dropoff_latitude float ,@dropoff_longitude
float', @model = @lmodel2, @passenger_count =@passenger_count,
@trip_distance=@trip_distance, @trip_time_in_secs=@trip_time_in_secs,
@pickup_latitude=@pickup_latitude, @pickup_longitude=@pickup_longitude,
```

```
@dropoff_latitude=@dropoff_latitude,  
@dropoff_longitude=@dropoff_longitude  
  WITH RESULT SETS ((Score float));  
END
```

2. Try it out, by providing the values manually.

Open a new **Query** window, and call the stored procedure, providing values for each of the parameters. The parameters represent feature columns used by the model and are required.

```
SQL  
  
EXEC [dbo].[RPredictSingleRow] @model = 'RTrainLogit_model',  
@passenger_count = 1,  
@trip_distance = 2.5,  
@trip_time_in_secs = 631,  
@pickup_latitude = 40.763958,  
@pickup_longitude = -73.973373,  
@dropoff_latitude = 40.782139,  
@dropoff_longitude = -73.977303
```

Or, use this shorter form supported for [parameters to a stored procedure](#):

```
SQL  
  
EXEC [dbo].[RPredictSingleRow] 'RTrainLogit_model', 1, 2.5, 631,  
40.763958, -73.973373, 40.782139, -73.977303
```

3. The results indicate that the probability of getting a tip is low (zero) on these top 10 trips, since all are single-passenger trips over a relatively short distance.

Conclusions

Now that you have learned to embed R code in stored procedures, you can extend these practices to build models of your own. The integration with Transact-SQL makes it much easier to deploy R models for prediction and to incorporate model retraining as part of an enterprise data workflow.

Next steps

In this article, you:

- ✓ Created and used stored procedures for batch scoring

✓ Created and used stored procedures for scoring a single row

For more information about R, see [R extension in SQL Server](#).

Tutorial: SQL development for R data scientists

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

In this tutorial for data scientists, learn how to build end-to-end solution for predictive modeling based on R feature support in either SQL Server 2016 or SQL Server 2017. This tutorial uses a [NYCTaxi_sample](#) database on SQL Server.

You use a combination of R code, SQL Server data, and custom SQL functions to build a classification model that indicates the probability that the driver might get a tip on a particular taxi trip. You also deploy your R model to SQL Server and use server data to generate scores based on the model.

This example can be extended to all kinds of real-life problems, such as predicting customer responses to sales campaigns, or predicting spending or attendance at events. Because the model can be invoked from a stored procedure, you can easily embed it in an application.

Because the walkthrough is designed to introduce R developers to R Services (In-Database), R is used wherever possible. However, this does not mean that R is necessarily the best tool for each task. In many cases, SQL Server might provide better performance, particularly for tasks such as data aggregation and feature engineering. Such tasks can particularly benefit from new features in SQL Server, such as memory optimized columnstore indexes. We try to point out possible optimizations along the way.

Prerequisites

- [SQL Server Machine Learning Services with R integration](#) or [SQL Server 2016 R Services](#)
- [Database permissions](#) and a SQL Server database user login
- [SQL Server Management Studio](#)
- [NYC Taxi demo database](#)
- An R IDE such as RStudio or the built-in RGUI tool included with R

We recommend that you do this walkthrough on a client workstation. You must be able to connect, on the same network, to a SQL Server computer with SQL Server and the R language enabled. For instructions on workstation configuration, see [Set up a data science client for R development](#).

Alternatively, you can run the walkthrough on a computer that has both SQL Server and an R development environment, but we don't recommend this configuration for a production environment. If you need to put client and server on the same computer, be sure to install a second set of Microsoft R libraries for sending R script from a "remote" client. Do not use the R libraries that are installed in the program files of the SQL Server instance. Specifically, if you are using one computer, you need the RevoScaleR library in both of these locations to support client and server operations.

- C:\Program Files\Microsoft\R Client\R_SERVER\library\RevoScaleR
- C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library\RevoScaleR

Additional R packages

This walkthrough requires several R libraries that are not installed by default as part of R Services (In-Database). You must install the packages both on the client where you develop the solution, and on the SQL Server computer where you deploy the solution.

On a client workstation

In your R environment, copy the following lines and execute the code in a Console window (Rgui or an IDE). Some packages also install required packages. In all, about 32 packages are installed. You must have an internet connection to complete this step.

R

```
# Install required R libraries, if they are not already installed.
if (!('ggmap' %in% rownames(installed.packages())))
{install.packages('ggmap')}
if (!('mapproj' %in% rownames(installed.packages())))
{install.packages('mapproj')}
if (!('ROCR' %in% rownames(installed.packages()))){install.packages('ROCR')}
if (!('RODBC' %in% rownames(installed.packages())))
{install.packages('RODBC')}
```

On the server

You have several options for installing packages on SQL Server. For example, SQL Server provides [R package management](#) feature that lets database administrators create a package repository and assign user the rights to install their own packages. However, if you are an administrator on the computer, you can install new packages using R, as long as you install to the correct library.

ⓘ Note

On the server, **do not** install to a user library even if prompted. If you install to a user library, the SQL Server instance cannot find or run the packages. For more information, see [Installing new R Packages on SQL Server](#).

1. On the SQL Server computer, open RGui.exe as an administrator. If you have installed SQL Server R Services using the defaults, Rgui.exe can be found in C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\bin\x64).
2. At an R prompt, run the following R commands:

```
R
install.packages("ggmap", lib=grep("Program Files", .libPaths(), value=TRUE)
[1])
install.packages("mapproj", lib=grep("Program Files", .libPaths(),
value=TRUE)[1])
install.packages("ROCR", lib=grep("Program Files", .libPaths(), value=TRUE)
[1])
install.packages("RODBC", lib=grep("Program Files", .libPaths(), value=TRUE)
[1])
```

This example uses the R grep function to search the vector of available paths and find the path that includes "Program Files". For more information, see <https://www.rdocumentation.org/packages/base/functions/grep>.

If you think the packages are already installed, check the list of installed packages by running `installed.packages()`.

Next steps

Explore and summarize the data

View and summarize SQL Server data using R (walkthrough)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This lesson introduces you to functions in the **RevoScaleR** package and steps you through the following tasks:

- ✓ Connect to SQL Server
- ✓ Define a query that has the data you need, or specify a table or view
- ✓ Define one or more compute contexts to use when running R code
- ✓ Optionally, define transformations that are applied to the data source while it is being read from the source

Define a SQL Server compute context

Run the following R statements in an R environment on the client workstation. This section assumes a [data science workstation with Microsoft R Client](#), because it includes all the RevoScaleR packages, as well as a basic, lightweight set of R tools. For example, you can use Rgui.exe to run the R script in this section.

1. If the **RevoScaleR** package is not already loaded, run this line of R code:

```
R
```

```
library("RevoScaleR")
```

The quotation marks are optional, in this case, though recommended.

If you get an error, make sure that your R development environment is using a library that includes the RevoScaleR package. Use a command such as `.libPaths()` to view the current library path.

2. Create the connection string for SQL Server and save it in an R variable, *connStr*.

You must change the placeholder "your_server_name" to a valid SQL Server instance name. For the server name, you might be able to use only the instance name, or you might need to fully qualify the name, depending on your network.

For SQL Server authentication, the connection syntax is as follows:

R

```
connStr <- "Driver=SQL
Server;Server=your_server_name;Database=nyctaxi_sample;Uid=your-sql-
login;Pwd=your-login-password"
```

For Windows authentication, the syntax is a bit different:

R

```
connStr <- "Driver=SQL
Server;Server=your_server_name;Database=nyctaxi_sample;Trusted_Connecti
on=True"
```

Generally, we recommend that you use Windows authentication where possible, to avoid saving passwords in your R code.

3. Define variables to use in creating a new *compute context*. After you create the compute context object, you can use it to run R code on the SQL Server instance.

R

```
sqlShareDir <- paste("C:\\AllShare\\", Sys.getenv("USERNAME"), sep="")
sqlWait <- TRUE
sqlConsoleOutput <- FALSE
```

- R uses a temporary directory when serializing R objects back and forth between your workstation and the SQL Server computer. You can specify the local directory that is used as *sqlShareDir*, or accept the default.
- Use *sqlWait* to indicate whether you want R to wait for results from the server. For a discussion of waiting versus non-waiting jobs, see [Distributed and parallel computing with RevoScaleR in Microsoft R](#).
- Use the argument *sqlConsoleOutput* to indicate that you don't want to see output from the R console.

4. You call the [RxInSqlServer](#) constructor to create the compute context object with the variables and connection strings already defined, and save the new object in the R variable *sqlcc*.

R

```
sqlcc <- RxInSqlServer(connectionString = connStr, shareDir =
sqlShareDir, wait = sqlWait, consoleOutput = sqlConsoleOutput)
```

5. By default, the compute context is local, so you need to explicitly set the *active* compute context.

```
R
```

```
rxSetComputeContext(sqlcc)
```

- [rxSetComputeContext](#) returns the previously active compute context invisibly so that you can use it
- [rxGetComputeContext](#) returns the active compute context

Note that setting a compute context only affects operations that use functions in the **RevoScaleR** package; the compute context does not affect the way that open-source R operations are performed.

Create a data source using RxSqlServer

When using the Microsoft R libraries like RevoScaleR and MicrosoftML, a *data source* is an object you create using RevoScaleR functions. The data source object specifies some set of data that you want to use for a task, such as model training or feature extraction. You can get data from a variety of sources including SQL Server. For the list of currently supported sources, see [RxDataSource](#).

Earlier you defined a connection string, and saved that information in an R variable. You can re-use that connection information to specify the data you want to get.

1. Save a SQL query as a string variable. The query defines the data for training the model.

```
R
```

```
sampleDataQuery <- "SELECT TOP 1000 tipped, fare_amount,  
passenger_count,trip_time_in_secs,trip_distance, pickup_datetime,  
dropoff_datetime, pickup_longitude, pickup_latitude, dropoff_longitude,  
dropoff_latitude FROM nyctaxi_sample"
```

We've used a TOP clause here to make things run faster, but the actual rows returned by the query can vary depending on order. Hence, your summary results might also be different from those listed below. Feel free to remove the TOP clause.

2. Pass the query definition as an argument to the [RxSqlServerData](#) function.

R

```
inDataSource <- RxSqlServerData(  
  sqlQuery = sampleDataQuery,  
  connectionString = connStr,  
  colClasses = c(pickup_longitude = "numeric", pickup_latitude =  
"numeric",  
  dropoff_longitude = "numeric", dropoff_latitude = "numeric"),  
  rowsPerRead=500  
)
```

- The argument *colClasses* specifies the column types to use when moving the data between SQL Server and R. This is important because SQL Server uses different data types than R, and more data types. For more information, see [R Libraries and Data Types](#).
- The argument *rowsPerRead* is important for managing memory usage and efficient computations. Most of the enhanced analytical functions in R Services (In-Database) process data in chunks and accumulate intermediate results, returning the final computations after all of the data has been read. By adding the *rowsPerRead* parameter, you can control how many rows of data are read into each chunk for processing. If the value of this parameter is too large, data access might be slow because you don't have enough memory to efficiently process such a large chunk of data. On some systems, setting *rowsPerRead* to an excessively small value can also provide slower performance.

3. At this point, you've created the *inDataSource* object, but it doesn't contain any data. The data is not pulled from the SQL query into the local environment until you run a function such as [rxImport](#) or [rxSummary](#).

However, now that you've defined the data objects, you can use it as the argument to other functions.

Use the SQL Server data in R summaries

In this section, you'll try out several of the functions provided in R Services (In-Database) that support remote compute contexts. By applying R functions to the data source, you can explore, summarize, and chart the SQL Server data.

1. Call the function [rxGetVarInfo](#) to get a list of the variables in the data source and their data types.

`rxGetVarInfo` is a handy function; you can call it on any data frame, or on a set of data in a remote data object, to get information such as the maximum and minimum values, the data type, and the number of levels in factor columns.

Consider running this function after any kind of data input, feature transformation, or feature engineering. By doing so, you can ensure that all the features you want to use in your model are of the expected data type and avoid errors.

```
R
```

```
rxGetVarInfo(data = inDataSource)
```

Results

```
R
```

```
Var 1: tipped, Type: integer  
Var 2: fare_amount, Type: numeric  
Var 3: passenger_count, Type: integer  
Var 4: trip_time_in_secs, Type: numeric, Storage: int64  
Var 5: trip_distance, Type: numeric  
Var 6: pickup_datetime, Type: character  
Var 7: dropoff_datetime, Type: character  
Var 8: pickup_longitude, Type: numeric  
Var 9: pickup_latitude, Type: numeric  
Var 10: dropoff_longitude, Type: numeric
```

2. Now, call the RevoScaleR function `rxSummary` to get more detailed statistics about individual variables.

`rxSummary` is based on the R `summary` function, but has some additional features and advantages. `rxSummary` works in multiple compute contexts and supports chunking. You can also use `rxSummary` to transform values, or summarize based on factor levels.

In this example, you summarize the fare amount based on the number of passengers.

```
R
```

```
start.time <- proc.time()  
rxSummary(~fare_amount:F(passenger_count,1,6), data = inDataSource)  
used.time <- proc.time() - start.time  
print(paste("It takes CPU Time=", round(used.time[1]+used.time[2],2), "  
seconds,  
Elapsed Time=", round(used.time[3],2),  
" seconds to summarize the inDataSource.", sep=""))
```

- The first argument to `rxSummary` specifies the formula or term to summarize by. Here, the `F()` function is used to convert the values in `passenger_count` into factors before summarizing. You also have to specify the minimum value (1) and maximum value (6) for the `passenger_count` factor variable.
- If you do not specify the statistics to output, by default `rxSummary` outputs Mean, StDev, Min, Max, and the number of valid and missing observations.
- This example also includes some code to track the time the function starts and completes, so that you can compare performance.

Results

If the `rxSummary` function runs successfully, you should see results like these, followed by a list of statistics by category.

```
R

rxSummary(formula = ~fare_amount:F(passenger_count, 1,6), data =
inDataSource)
Data: inDataSource (RxSqlServerData Data Source)
Number of valid observations: 1000
```

Bonus exercise on big data

Try defining a new query string with all the rows. We recommend you set up a new data source object for this experiment. You might also try changing the `rowsToRead` parameter to see how it affects throughput.

```
R

bigDataQuery <- "SELECT tipped, fare_amount,
passenger_count,trip_time_in_secs,trip_distance, pickup_datetime,
dropoff_datetime, pickup_longitude, pickup_latitude, dropoff_longitude,
dropoff_latitude FROM nyctaxi_sample"

bigDataSource <- RxSqlServerData(
  sqlQuery = bigDataQuery,
  connectionString = connStr,
  colClasses = c(pickup_longitude = "numeric", pickup_latitude =
"numeric",
  dropoff_longitude = "numeric", dropoff_latitude = "numeric"),
  rowsPerRead=500
)

start.time <- proc.time()
rxSummary(~fare_amount:F(passenger_count,1,6), data = bigDataSource)
used.time <- proc.time() - start.time
```



```
print(paste("It takes CPU Time=", round(used.time[1]+used.time[2],2),"
seconds,
  Elapsed Time=", round(used.time[3],2),
  " seconds to summarize the inDataSource.", sep=""))
```

Tip

While this is running, you can use a tool like **Process Explorer** or SQL Profiler to see how the connection is made and the R code is run using SQL Server services.

Next steps

Create graphs and plots using R

Create graphs and plots using SQL and R (walkthrough)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

In this part of the walkthrough, you learn techniques for generating plots and maps using R with SQL Server data. You create a simple histogram and then develop a more complex map plot.

Prerequisites

This step assumes an ongoing R session based on previous steps in this walkthrough. It uses the connection strings and data source objects created in those steps. The following tools and packages are used to run the script.

- Rgui.exe to run R commands
- Management Studio to run T-SQL
- googMap
- ggmap package
- mapproj package

Create a histogram

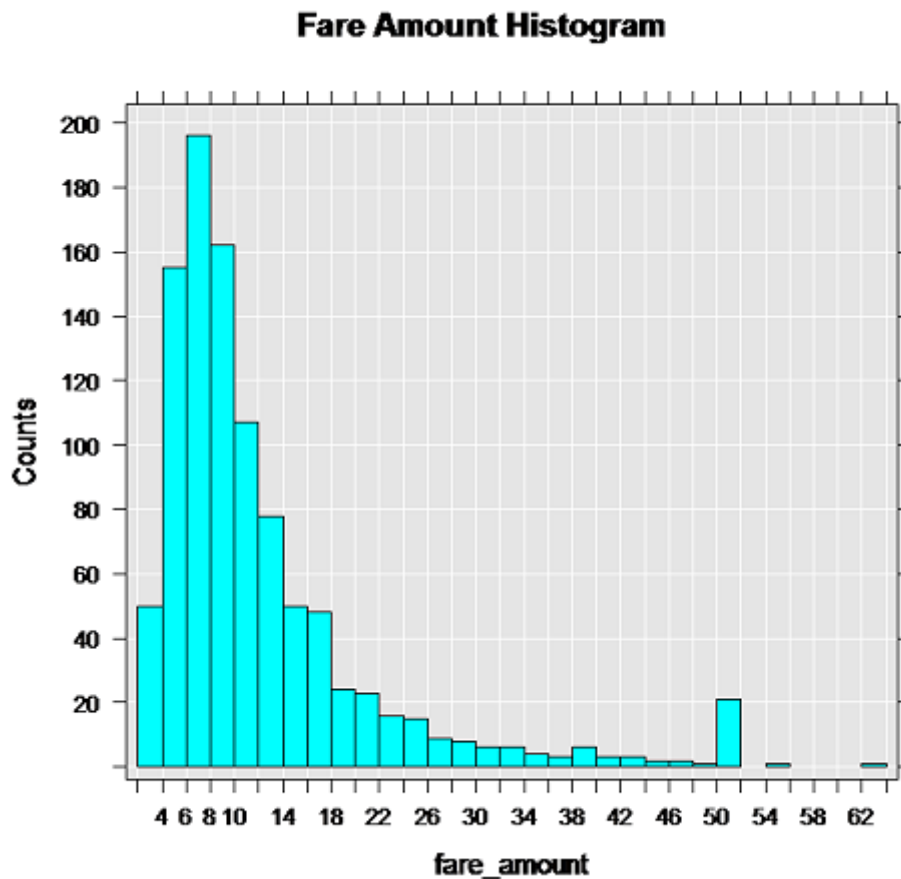
1. Generate the first plot, using the [rxHistogram](#) function. The rxHistogram function provides functionality similar to that in open-source R packages, but can run in a remote execution context.

R

```
# Plot fare amount on SQL Server and return the plot
start.time <- proc.time()
rxHistogram(~fare_amount, data = inDataSource, title = "Fare Amount
Histogram")
used.time <- proc.time() - start.time
print(paste("It takes CPU Time=", round(used.time[1]+used.time[2],2), "
seconds, Elapsed Time=", round(used.time[3],2), " seconds to generate
plot.", sep=""))
```

2. The image is returned in the R graphics device for your development environment. For example, in RStudio, click the **Plot** window. In R Tools for Visual Studio, a

separate graphics window is opened.



⚠ Note

Does your graph look different?

That's because *inDataSource* uses only the top 1000 rows. The ordering of rows using TOP is non-deterministic in the absence of an ORDER BY clause, so it's expected that the data and the resulting graph might vary. This particular image was generated using about 10,000 rows of data. We recommend that you experiment with different numbers of rows to get different graphs, and note how long it takes to return the results in your environment.

Create a map plot

Typically, database servers block Internet access. This can be inconvenient when using R packages that need to download maps or other images to generate plots. However, there is a workaround that you might find useful when developing your own applications. Basically, you generate the map representation on the client, and then overlay on the map the points that are stored as attributes in the SQL Server table.

1. Define the function that creates the R plot object. The custom function *mapPlot* creates a scatter plot that uses the taxi pickup locations, and plots the number of rides that started from each location. It uses the **ggplot2** and **ggmap** packages, which should already be [installed and loaded](#).

R

```
mapPlot <- function(inDataSource, googMap){
  library(ggmap)
  library(mapproj)
  ds <- rxImport(inDataSource)
  p <- ggmap(googMap)+
  geom_point(aes(x = pickup_longitude, y =pickup_latitude ), data=ds,
alpha =.5,
color="darkred", size = 1.5)
  return(list(myplot=p))
}
```

- The *mapPlot* function takes two arguments: an existing data object, which you defined earlier using *RxSqlServerData*, and the map representation passed from the client.
- In the line beginning with the *ds* variable, *rxImport* is used to load into memory data from the previously created data source, *inDataSource*. (That data source contains only 1000 rows; if you want to create a map with more data points, you can substitute a different data source.)
- Whenever you use open-source R functions, data must be loaded into data frames in local memory. However, by calling the *rxImport* function, you can run in the memory of the remote compute context.

2. Change the compute context to local, and load the libraries required for creating the maps.

R

```
rxSetComputeContext("local")
library(ggmap)
library(mapproj)
gc <- geocode("Times Square", source = "google")
googMap <- get_googlemap(center = as.numeric(gc), zoom = 12, maptype =
'roadmap', color = 'color');
```

- The `gc` variable stores a set of coordinates for Times Square, NY.
- The line beginning with `googmap` generates a map with the specified coordinates at the center.

3. Switch to the SQL Server compute context, and render the results, by wrapping the plot function in `rxExec` as shown here. The `rxExec` function is part of the **RevoScaleR** package, and supports execution of arbitrary R functions in a remote compute context.

R

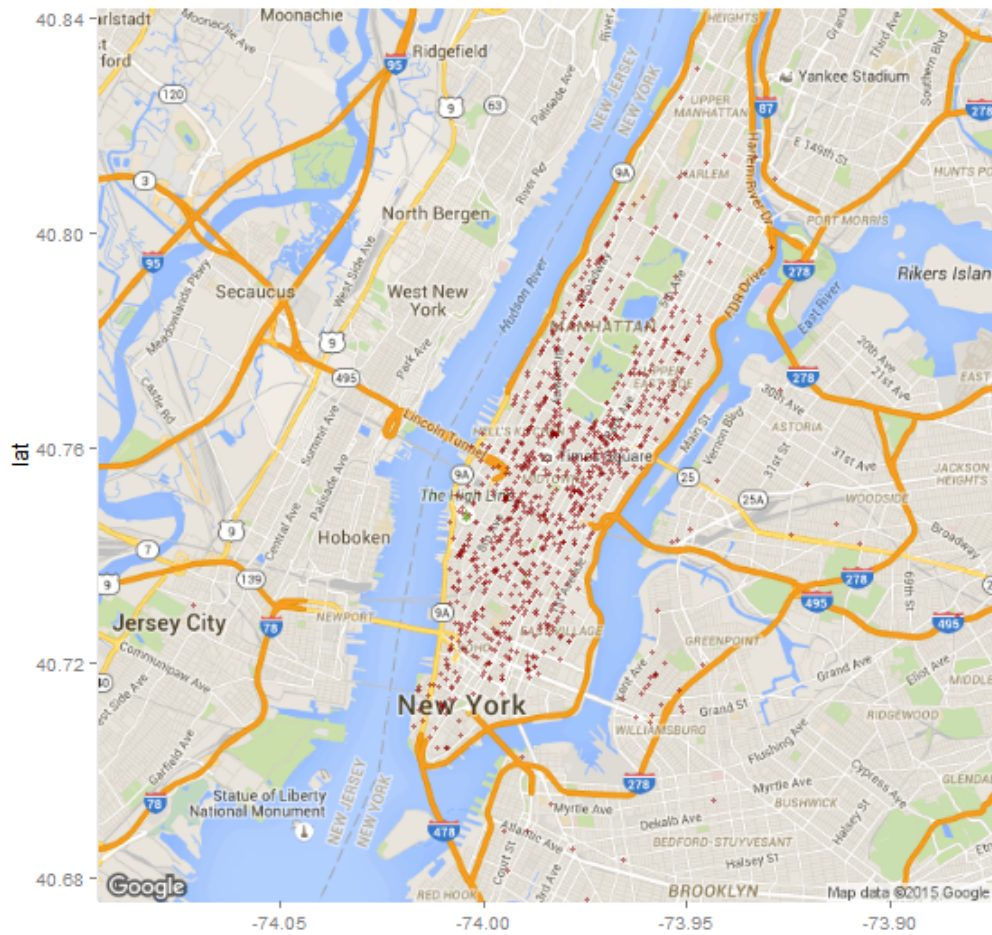
```
rxSetComputeContext(sqlcc)
myplots <- rxExec(mapPlot, inDataSource, googMap, timesToRun = 1)
plot(myplots[[1]][["myplot"]]);
```

- The map data in `googMap` is passed as an argument to the remotely executed function `mapPlot`. Because the maps were generated in your local environment, they must be passed to the function in order to create the plot in the context of SQL Server.
- When the line beginning with `plot` runs, the rendered data is serialized back to the local R environment so that you can view it in your R client.

ⓘ Note

If you are using SQL Server in an Azure virtual machine, you might get an error at this point. An error occurs when the default firewall rule in Azure blocks network access by R code. For details on how to fix this error, see [Installing Machine Learning \(R\) Services on an Azure VM](#).

4. The following image shows the output plot. The taxi pickup locations are added to the map as red dots. Your image might look different, depending how many locations are in the data source you used.



Next steps


Create data features using R and SQL

Create data features using R and SQL Server (walkthrough)



Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Data engineering is an important part of machine learning. Data often requires transformation before you can use it for predictive modeling. If the data does not have the features you need, you can engineer them from existing values.

For this modeling task, rather than using the raw latitude and longitude values of the pickup and drop-off location, you'd like to have the distance in miles between the two locations. To create this feature, you compute the direct linear distance between two points, by using the [haversine formula](#) .

In this step, learn two different methods for creating a feature from data:

-  Using a custom R function
-  Using a custom T-SQL function in Transact-SQL

The goal is to create a new SQL Server set of data that includes the original columns plus the new numeric feature, *direct_distance*.

Prerequisites

This step assumes an ongoing R session based on previous steps in this walkthrough. It uses the connection strings and data source objects created in those steps. The following tools and packages are used to run the script.

- Rgui.exe to run R commands
- Management Studio to run T-SQL

Featurization using R

The R language is well-known for its rich and varied statistical libraries, but you still might need to create custom data transformations.

First, let's do it the way R users are accustomed to: get the data onto your laptop, and then run a custom R function, *ComputeDist*, which calculates the linear distance between two points specified by latitude and longitude values.

1. Remember that the data source object you created earlier gets only the top 1000 rows. So let's define a query that gets all the data.

R

```
bigQuery <- "SELECT tipped, fare_amount,
passenger_count,trip_time_in_secs,trip_distance, pickup_datetime,
dropoff_datetime, pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude FROM nyctaxi_sample";
```

2. Create a new data source object using the query.

R

```
featureDataSource <- RxSqlServerData(sqlQuery = bigQuery,colClasses =
c(pickup_longitude = "numeric", pickup_latitude = "numeric",
dropoff_longitude = "numeric", dropoff_latitude = "numeric",
passenger_count = "numeric", trip_distance = "numeric",
trip_time_in_secs = "numeric", direct_distance = "numeric"),
connectionString = connStr);
```

- [RxSqlServerData](#) can take either a query consisting of a valid SELECT query, provided as the argument to the *sqlQuery* parameter, or the name of a table object, provided as the *table* parameter.
- If you want to sample data from a table, you must use the *sqlQuery* parameter, define sampling parameters using the T-SQL TABLESAMPLE clause, and set the *rowBuffering* argument to FALSE.

3. Run the following code to create the custom R function. ComputeDist takes in two pairs of latitude and longitude values, and calculates the linear distance between them, returning the distance in miles.

R

```
env <- new.env();
env$ComputeDist <- function(pickup_long, pickup_lat, dropoff_long,
dropoff_lat){
  R <- 6371/1.609344 #radius in mile
  delta_lat <- dropoff_lat - pickup_lat
  delta_long <- dropoff_long - pickup_long
  degrees_to_radians = pi/180.0
  a1 <- sin(delta_lat/2*degrees_to_radians)
  a2 <- as.numeric(a1)^2
  a3 <- cos(pickup_lat*degrees_to_radians)
  a4 <- cos(dropoff_lat*degrees_to_radians)
  a5 <- sin(delta_long/2*degrees_to_radians)
  a6 <- as.numeric(a5)^2
```



```

a <- a2+a3*a4*a6
c <- 2*atan2(sqrt(a),sqrt(1-a))
d <- R*c
return (d)
}

```

- The first line defines a new environment. In R, an environment can be used to encapsulate name spaces in packages and such. You can use the `search()` function to view the environments in your workspace. To view the objects in a specific environment, type `ls(<envname>)`.
- The lines beginning with `$env.ComputeDist` contain the code that defines the haversine formula, which calculates the *great-circle distance* between two points on a sphere.

4. Having defined the function, you apply it to the data to create a new feature column, *direct_distance*. but before you run the transformation, change the compute context to local.

```

R

rxSetComputeContext("local");

```

5. Call the `rxDataStep` function to get the feature engineering data, and apply the `env$ComputeDist` function to the data in memory.

```

R

start.time <- proc.time();

changed_ds <- rxDataStep(inData = featureDataSource,
  transforms =
  list(direct_distance=ComputeDist(pickup_longitude,pickup_latitude,
  dropoff_longitude, dropoff_latitude),
  tipped = "tipped", fare_amount = "fare_amount", passenger_count =
  "passenger_count",
  trip_time_in_secs = "trip_time_in_secs",
  trip_distance="trip_distance",
  pickup_datetime = "pickup_datetime", dropoff_datetime =
  "dropoff_datetime"),
  transformEnvir = env,
  rowsPerRead=500,
  reportProgress = 3);

used.time <- proc.time() - start.time;
print(paste("It takes CPU Time=", round(used.time[1]+used.time[2],2), "
seconds, Elapsed Time=", round(used.time[3],2), " seconds to generate
features.", sep=""));

```

- The `rxDataStep` function supports various methods for modifying data in place. For more information, see this article: [How to transform and subset data in Microsoft R](#)

However, a couple of points worth noting regarding `rxDataStep`:

In other data sources, you can use the arguments `varsToKeep` and `varsToDrop`, but these are not supported for SQL Server data sources. Therefore, in this example, we've used the `transforms` argument to specify both the pass-through columns and the transformed columns. Also, when running in a SQL Server compute context, the `inData` argument can only take a SQL Server data source.

The preceding code can also produce a warning message when run on larger data sets. When the number of rows times the number of columns being created exceeds a set value (the default is 3,000,000), `rxDataStep` returns a warning, and the number of rows in the returned data frame will be truncated. To remove the warning, you can modify the `maxRowsByCols` argument in the `rxDataStep` function. However, if `maxRowsByCols` is too large, you might experience problems when loading the data frame into memory.

6. Optionally, you can call `rxGetVarInfo` to inspect the schema of the transformed data source.

```
R
```

```
rxGetVarInfo(data = changed_ds);
```

Featurization using Transact-SQL

In this exercise, learn how to accomplish the same task using SQL functions instead of custom R functions.

Switch to [SQL Server Management Studio](#) or another query editor to run the T-SQL script.

1. Use a SQL function, named `fnCalculateDistance`. The function should already exist in the `NYCTaxi_Sample` database. In Object Explorer, verify the function exists by navigating this path: Databases > NYCTaxi_Sample > Programmability > Functions > Scalar-valued Functions > `dbo.fnCalculateDistance`.

If the function does not exist, use SQL Server Management Studio to generate the function in the `NYCTaxi_Sample` database.

SQL

```
CREATE FUNCTION [dbo].[fnCalculateDistance] (@Lat1 float, @Long1 float,
@Lat2 float, @Long2 float)
-- User-defined function calculates the direct distance between two
geographical coordinates.
RETURNS decimal(28, 10)
AS
BEGIN
    DECLARE @distance decimal(28, 10)
    -- Convert to radians
    SET @Lat1 = @Lat1 / 57.2958
    SET @Long1 = @Long1 / 57.2958
    SET @Lat2 = @Lat2 / 57.2958
    SET @Long2 = @Long2 / 57.2958
    -- Calculate distance
    SET @distance = (SIN(@Lat1) * SIN(@Lat2)) + (COS(@Lat1) * COS(@Lat2)
* COS(@Long2 - @Long1))
    --Convert to miles
    IF @distance <> 0
    BEGIN
        SET @distance = 3958.75 * ATAN(SQRT(1 - POWER(@distance, 2)) /
@distance);
    END
    RETURN @distance
END
```

2. In Management Studio, in a new query window, run the following Transact-SQL statement from any application that supports Transact-SQL to see how the function works.

SQL

```
USE nyctaxi_sample
GO

SELECT tipped, fare_amount,
passenger_count,trip_time_in_secs,trip_distance, pickup_datetime,
dropoff_datetime,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude) as direct_distance,
pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
FROM nyctaxi_sample
```

3. To insert values directly into a new table (you have to create it first), you can add an **INTO** clause specifying the table name.

SQL

```

USE nyctaxi_sample
GO

SELECT tipped, fare_amount, passenger_count, trip_time_in_secs,
trip_distance, pickup_datetime, dropoff_datetime,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude) as direct_distance,
pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
INTO NewFeatureTable
FROM nyctaxi_sample

```

4. You can also call the SQL function from R code. Switch back to Rgui and store the SQL featurization query in an R variable.

```

R

featureEngineeringQuery = "SELECT tipped, fare_amount, passenger_count,
    trip_time_in_secs,trip_distance, pickup_datetime, dropoff_datetime,
    dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
    dropoff_latitude, dropoff_longitude) as direct_distance,
    pickup_latitude, pickup_longitude, dropoff_latitude,
    dropoff_longitude
    FROM nyctaxi_sample
    tablesample (1 percent) repeatable (98052)"

```

Tip

This query has been modified to get a smaller sample of data, to make this walkthrough faster. You can remove the TABLESAMPLE clause if you want to get all the data; however, depending on your environment, it might not be possible to load the full dataset into R, resulting in an error.

5. Use the following lines of code to call the Transact-SQL function from your R environment and apply it to the data defined in *featureEngineeringQuery*.

```

R

featureDataSource = RxSqlServerData(sqlQuery = featureEngineeringQuery,
    colClasses = c(pickup_longitude = "numeric", pickup_latitude =
"numeric",
    dropoff_longitude = "numeric", dropoff_latitude = "numeric",
    passenger_count = "numeric", trip_distance = "numeric",
    trip_time_in_secs = "numeric", direct_distance = "numeric"),
    connectionString = connStr)

```

6. Now that the new feature is created, call `rxGetVarsInfo` to create a summary of the data in the feature table.

```
R
```

```
rxGetVarInfo(data = featureDataSource)
```

Results

```
R
```

```
Var 1: tipped, Type: integer  
Var 2: fare_amount, Type: numeric  
Var 3: passenger_count, Type: numeric  
Var 4: trip_time_in_secs, Type: numeric  
Var 5: trip_distance, Type: numeric  
Var 6: pickup_datetime, Type: character  
Var 7: dropoff_datetime, Type: character  
Var 8: direct_distance, Type: numeric  
Var 9: pickup_latitude, Type: numeric  
Var 10: pickup_longitude, Type: numeric  
Var 11: dropoff_latitude, Type: numeric  
Var 12: dropoff_longitude, Type: numeric
```

ⓘ Note

In some cases, you might get an error like this one: *The EXECUTE permission was denied on the object 'fnCalculateDistance'* If so, make sure that the login you are using has permissions to run scripts and create objects on the database, not just on the instance. Check the schema for the object, `fnCalculateDistance`. If the object was created by the database owner, and your login belongs to the role `db_datareader`, you need to give the login explicit permissions to run the script.

Comparing R functions and SQL functions

Remember this piece of code used to time the R code?

```
R
```

```
start.time <- proc.time()  
<your code here>  
used.time <- proc.time() - start.time  
print(paste("It takes CPU Time=", round(used.time[1]+used.time[2],2),"
```

```
seconds, Elapsed Time=", round(used.time[3],2), " seconds to generate features.", sep="")
```

You can try using this with the SQL custom function example to see how long the data transformation takes when calling a SQL function. Also, try switching compute contexts with `rxSetComputeContext` and compare the timings.

Your times might vary significantly, depending on your network speed, and your hardware configuration. In the configurations we tested, the Transact-SQL function approach was faster than using a custom R function. Therefore, we've use the Transact-SQL function for these calculations in subsequent steps.

Tip

Very often, feature engineering using Transact-SQL will be faster than R. For example, T-SQL includes fast windowing and ranking functions that can be applied to common data science calculations such as rolling moving averages and n -tiles. Choose the most efficient method based on your data and task.

Next steps

[Build an R model and save to SQL](#)

Build an R model and save to SQL Server (walkthrough)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

In this step, learn how to build a machine learning model and save the model in SQL Server. By saving a model, you can call it directly from Transact-SQL code, using the system stored procedure, [sp_execute_external_script](#) or the [PREDICT \(T-SQL\) function](#).

Prerequisites

This step assumes an ongoing R session based on previous steps in this walkthrough. It uses the connection strings and data source objects created in those steps. The following tools and packages are used to run the script.

- Rgui.exe to run R commands
- Management Studio to run T-SQL
- ROCR package
- RODBC package

Create a stored procedure to save models

This step uses a stored procedure to save a trained model to SQL Server. Creating a stored procedure to perform this operation makes the task easier.

Run the following T-SQL code in a query windows in Management Studio to create the stored procedure.

SQL

```
USE [NYCTaxi_Sample]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P' AND name =
'PersistModel')
    DROP PROCEDURE PersistModel
GO
```

```
CREATE PROCEDURE [dbo].[PersistModel] @m nvarchar(max)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    insert into nyc_taxi_models (model) values
    (convert(varbinary(max),@m,2))
END
GO
```

ⓘ Note

If you get an error, make sure that your login has permission to create objects. You can grant explicit permissions to create objects by running a T-SQL statement like this: `exec sp_addrolemember 'db_owner', '<user_name>'`.

Create a classification model using rxLogit

The model is a binary classifier that predicts whether the taxi driver is likely to get a tip on a particular ride or not. You'll use the data source you created in the previous lesson to train the tip classifier, using logistic regression.

1. Call the `rxLogit` function, included in the **RevoScaleR** package, to create a logistic regression model.

R

```
system.time(logitObj <- rxLogit(tipped ~ passenger_count +
trip_distance + trip_time_in_secs + direct_distance, data =
featureDataSource));
```

The call that builds the model is enclosed in the `system.time` function. This lets you get the time required to build the model.

2. After you build the model, you can inspect it using the `summary` function, and view the coefficients.

R

```
summary(logitObj);
```

Results

R

```
*Logistic Regression Results for: tipped ~ passenger_count +
trip_distance + trip_time_in_secs +
direct_distance*
>Data: featureDataSource (RxSqlServerData Data Source)*
*Dependent variable(s): tipped*
*Total independent variables: 5*
*Number of valid observations: 17068*
*Number of missing observations: 0*
*-2\*LogLikelihood: 23540.0602 (Residual deviance on 17063 degrees of
freedom)*
*Coefficients:*
*Estimate Std. Error z value Pr(>|z|)*
*(Intercept)      -2.509e-03  3.223e-02  -0.078  0.93793*
*passenger_count  -5.753e-02  1.088e-02  -5.289  1.23e-07 \*\*\*\*
*trip_distance    -3.896e-02  1.466e-02  -2.658  0.00786 \*\*\*
*trip_time_in_secs  2.115e-04  4.336e-05   4.878  1.07e-06 \*\*\*\*
*direct_distance   6.156e-02  2.076e-02   2.966  0.00302 \*\*\*\*
*___*
*Signif. codes:  0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1*
*Condition number of final variance-covariance matrix: 48.3933*
*Number of iterations: 4*
```

Use the logistic regression model for scoring

Now that the model is built, you can use to predict whether the driver is likely to get a tip on a particular drive or not.

1. First, use the `RxSqlServerData` function to define a data source object for storing the scoring result.

R

```
scoredOutput <- RxSqlServerData(
  connectionString = connStr,
  table = "taxiScoreOutput" )
```

- To make this example simpler, the input to the logistic regression model is the same feature data source (`sql_feature_ds`) that you used to train the model. More typically, you might have some new data to score with, or you might have set aside some data for testing vs. training.
- The prediction results will be saved in the table, `taxiScoreOutput`. Notice that the schema for this table is not defined when you create it using `RxSqlServerData`. The schema is obtained from the `rxPredict` output.

- To create the table that stores the predicted values, the SQL login running the rxSqlServer data function must have DDL privileges in the database. If the login cannot create tables, the statement fails.

2. Call the `rxPredict` function to generate results.

```
R  
  
rxPredict(modelObject = logitObj,  
          data = featureDataSource,  
          outData = scoredOutput,  
          predVarNames = "Score",  
          type = "response",  
          writeModelVars = TRUE, overwrite = TRUE)
```

If the statement succeeds, it should take some time to run. When complete, you can open SQL Server Management Studio and verify that the table was created and that it contains the Score column and other expected output.

Plot model accuracy

To get an idea of the accuracy of the model, you can use the `rxRoc` function to plot the Receiver Operating Curve. Because `rxRoc` is one of the new functions provided by the `RevoScaleR` package that supports remote compute contexts, you have two options:

- You can use the `rxRoc` function to execute the plot in the remote compute context and then return the plot to your local client.
- You can also import the data to your R client computer, and use other R plotting functions to create the performance graph.

In this section, you'll experiment with both techniques.

Execute a plot in the remote (SQL Server) compute context

1. Call the function `rxRoc` and provide the data defined earlier as input.

```
R  
  
scoredOutput = rxImport(scoredOutput);  
rxRoc(actualVarName= "tipped", predVarNames = "Score", scoredOutput);
```

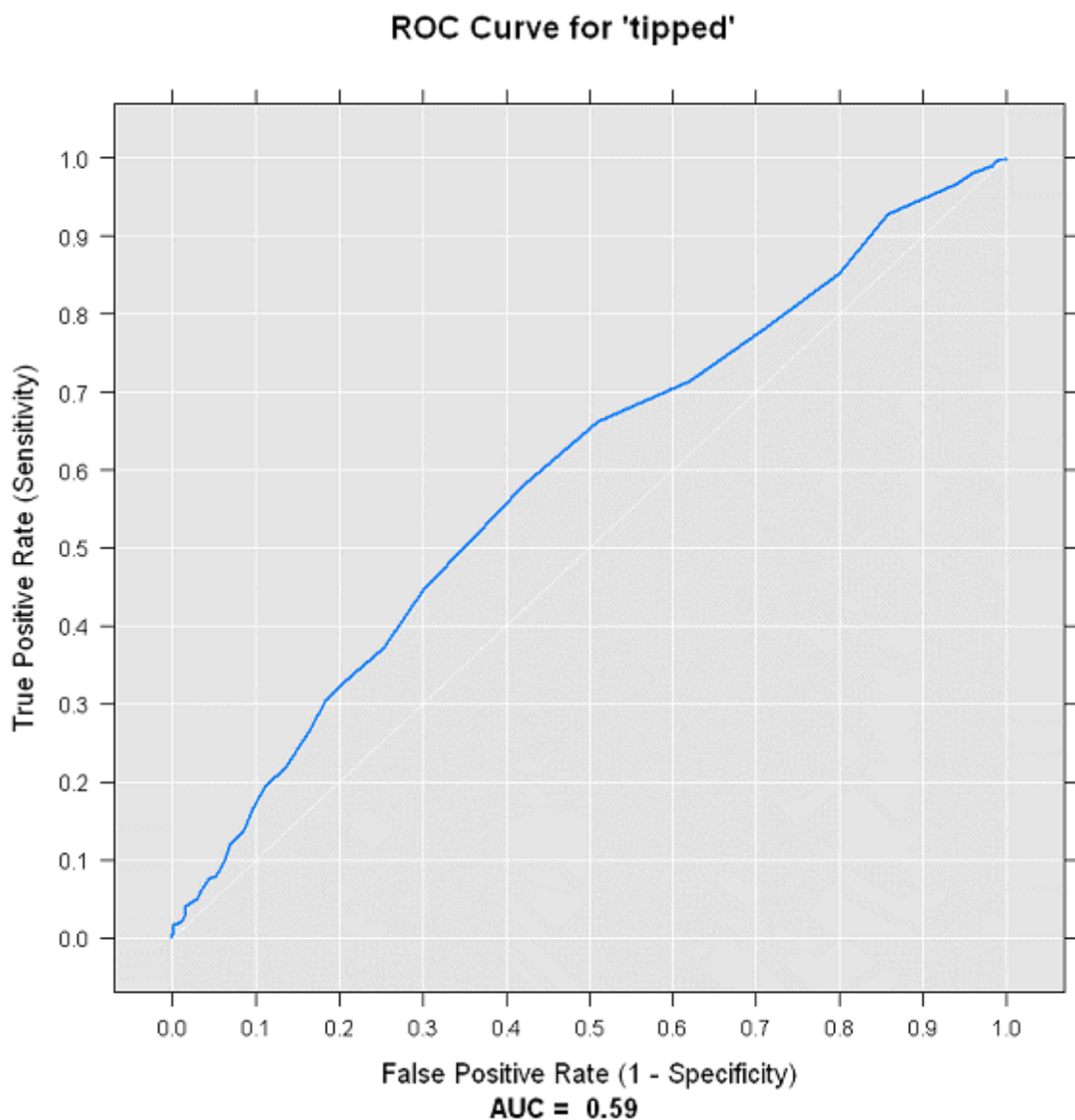
This call returns the values used in computing the ROC chart. The label column is *tipped*, which has the actual results you are trying to predict, while the *Score* column has the prediction.

2. To actually plot the chart, you can save the ROC object and then draw it with the plot function. The graph is created on the remote compute context, and returned to your R environment.

```
R

scoredOutput = rxImport(scoredOutput);
rocObjectOut <- rxRoc(actualVarName= "tipped", predVarNames = "Score",
scoredOutput);
plot(rocObjectOut);
```

View the graph by opening the R graphics device, or by clicking the **Plot** window in RStudio.



Create the plots in the local compute context using data from SQL Server

You can verify the compute context is local by running `rxGetComputeContext()` at the command prompt. The return value should be "RxLocalSeq Compute Context".

1. For the local compute context, the process is much the same. You use the `rxImport` function to bring the specified data into your local R environment.

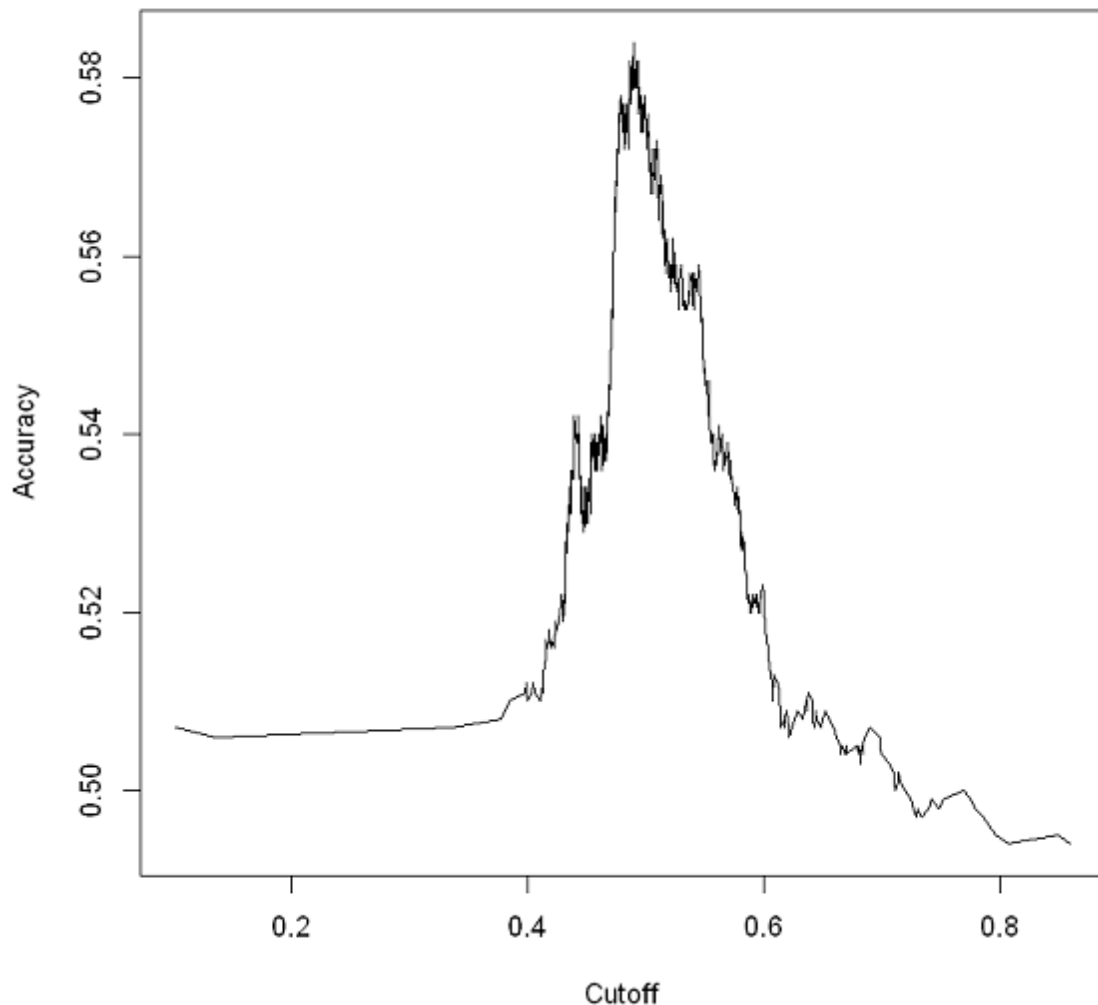
```
R
scoredOutput = rxImport(scoredOutput)
```

2. Using the data in local memory, you load the **ROCR** package, and use the prediction function from that package to create some new predictions.

```
R
library('ROCR');
pred <- prediction(scoredOutput$Score, scoredOutput$tipped);
```

3. Generate a local plot, based on the values stored in the output variable `pred`.

```
R
acc.perf = performance(pred, measure = 'acc');
plot(acc.perf);
ind = which.max( slot(acc.perf, 'y.values')[[1]] );
acc = slot(acc.perf, 'y.values')[[1]][ind];
cutoff = slot(acc.perf, 'x.values')[[1]][ind];
```



ⓘ Note

Your charts might look different from these, depending on how many data points you used.

Deploy the model

After you have built a model and ascertained that it is performing well, you probably want to deploy it to a site where users or people in your organization can make use of the model, or perhaps retrain and recalibrate the model on a regular basis. This process is sometimes called *operationalizing* a model. In SQL Server, operationalization is achieved by embedding R code in a stored procedure. Because code resides in the procedure, it can be called from any application that can connect to SQL Server.

Before you can call the model from an external application, you must save the model to the database used for production. Trained models are stored in binary form, in a single

column of type **varbinary(max)**.

A typical deployment workflow consists of the following steps:

1. Serialize the model into a hexadecimal string
2. Transmit the serialized object to the database
3. Save the model in a **varbinary(max)** column

In this section, learn how to use a stored procedure to persist the model and make it available for predictions. The stored procedure used in this section is `PersistModel`. The definition of `PersistModel` is in [Prerequisites](#).

1. Switch back to your local R environment if you are not already using it, serialize the model, and save it in a variable.

```
R  
  
rxSetComputeContext("local");  
modelbin <- serialize(logitObj, NULL);  
modelbinstr=paste(modelbin, collapse="");
```

2. Open an ODBC connection using **RODBC**. You can omit the call to `RODBC` if you already have the package loaded.

```
R  
  
library(RODBC);  
conn <- odbcDriverConnect(connStr);
```

3. Call the `PersistModel` stored procedure on SQL Server to transmit the serialized object to the database and store the binary representation of the model in a column.

```
R  
  
q <- paste("EXEC PersistModel @m='", modelbinstr, "'", sep="");  
sqlQuery (conn, q);
```

4. Use Management Studio to verify the model exists. In Object Explorer, right-click on the **nyc_taxi_models** table and click **Select Top 1000 Rows**. In Results, you should see a binary representation in the **models** column.

Saving a model to a table requires only an `INSERT` statement. However, it's often easier when wrapped in a stored procedure, such as *PersistModel*.

Next steps

In the next and final lesson, learn how to perform scoring against the saved model using Transact-SQL.

[Deploy the R model and use in SQL](#)

Deploy the R model and use it in SQL Server (walkthrough)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

In this lesson, learn how to deploy R models in a production environment by calling a trained model from a stored procedure. You can invoke the stored procedure from R or any application programming language that supports Transact-SQL (such as C#, Java, Python, and so forth) and use the model to make predictions on new observations.

This article demonstrates the two most common ways to use a model in scoring:

- ✓ **Batch scoring mode** generates multiple predictions
- ✓ **Individual scoring mode** generates predictions one at a time

Batch scoring

Create a stored procedure, *PredictTipBatchMode*, that generates multiple predictions, passing a SQL query or table as input. A table of results is returned, which you might insert directly into a table or write to a file.

- Gets a set of input data as a SQL query
- Calls the trained logistic regression model that you saved in the previous lesson
- Predicts the probability that the driver gets any non-zero tip

1. In Management Studio, open a new query window and run the following T-SQL script to create the PredictTipBatchMode stored procedure.

SQL

```
USE [NYCTaxi_Sample]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P' AND name =
'PredictTipBatchMode')
DROP PROCEDURE v
GO
```



```

CREATE PROCEDURE [dbo].[PredictTipBatchMode] @input nvarchar(max)
AS
BEGIN
    DECLARE @lmodel2 varbinary(max) = (SELECT TOP 1 model FROM
nyc_taxi_models);
    EXEC sp_execute_external_script @language = N'R',
        @script = N'
            mod <- unserialize(as.raw(model));
            print(summary(mod))
            OutputDataSet<-rxPredict(modelObject = mod,
                data = InputDataSet,
                outData = NULL,
                predVarNames = "Score", type = "response",
                writeModelVars = FALSE, overwrite = TRUE);
            str(OutputDataSet)
            print(OutputDataSet)',
        @input_data_1 = @input,
        @params = N'@model varbinary(max)',
        @model = @lmodel2
    WITH RESULT SETS ((Score float));
END

```

- You use a SELECT statement to call the stored model from a SQL table. The model is retrieved from the table as `varbinary(max)` data, stored in the SQL variable `@lmodel2`, and passed as the parameter `mod` to the system stored procedure `sp_execute_external_script`.
- The data used as inputs for scoring is defined as a SQL query and stored as a string in the SQL variable `@input`. As data is retrieved from the database, it is stored in a data frame called `InputDataSet`, which is just the default name for input data to the `sp_execute_external_script` procedure; you can define another variable name if needed by using the parameter `@input_data_1_name`.
- To generate the scores, the stored procedure calls the `rxPredict` function from the **RevoScaleR** library.
- The return value, `Score`, is the probability, given the model, that driver gets a tip. Optionally, you could easily apply some kind of filter to the returned values to categorize the return values into "tip" and "no tip" groups. For example, a probability of less than 0.5 would mean a tip is unlikely.

2. To call the stored procedure in batch mode, you define the query required as input to the stored procedure. Below is the SQL query, which you can run in SSMS to verify that it works.

SQL

```

SELECT TOP 10
  a.passenger_count AS passenger_count,
  a.trip_time_in_secs AS trip_time_in_secs,
  a.trip_distance AS trip_distance,
  a.dropoff_datetime AS dropoff_datetime,
  dbo.fnCalculateDistance( pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude) AS direct_distance
FROM
  (SELECT medallion, hack_license, pickup_datetime,
passenger_count,trip_time_in_secs,trip_distance, dropoff_datetime,
pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
FROM nyctaxi_sample)a
LEFT OUTER JOIN
( SELECT medallion, hack_license, pickup_datetime
FROM nyctaxi_sample  tablesample (1 percent) repeatable (98052) )b
ON a.medallion=b.medallion
AND a.hack_license=b.hack_license
AND a.pickup_datetime=b.pickup_datetime
WHERE b.medallion is null

```

3. Use this R code to create the input string from the SQL query:

```

R

input <- "N'SELECT TOP 10 a.passenger_count AS passenger_count,
a.trip_time_in_secs AS trip_time_in_secs, a.trip_distance AS
trip_distance, a.dropoff_datetime AS dropoff_datetime,
dbo.fnCalculateDistance(pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude) AS direct_distance FROM (SELECT
medallion, hack_license, pickup_datetime,
passenger_count,trip_time_in_secs,trip_distance, dropoff_datetime,
pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude
FROM nyctaxi_sample)a LEFT OUTER JOIN ( SELECT medallion, hack_license,
pickup_datetime FROM nyctaxi_sample  tablesample (1 percent) repeatable
(98052) )b ON a.medallion=b.medallion AND
a.hack_license=b.hack_license AND a.pickup_datetime=b.pickup_datetime
WHERE b.medallion is null";
q <- paste("EXEC PredictTipBatchMode @input = ", input, sep="");

```

4. To run the stored procedure from R, call the `sqlQuery` method of the `RODBC` package and use the SQL connection `conn` that you defined earlier:

```

R

sqlQuery (conn, q);

```

If you get an ODBC error, check for syntax errors and whether you have the right number of quotation marks.

If you get a permissions error, make sure the login has the ability to execute the stored procedure.

Single row scoring

Individual scoring mode generates predictions one at a time, passing a set of individual values to the stored procedure as input. The values correspond to features in the model, which the model uses to create a prediction, or generate another result such as a probability value. You can then return that value to the application, or user.

When calling the model for prediction on a row-by-row basis, you pass a set of values that represent features for each individual case. The stored procedure then returns a single prediction or probability.

The stored procedure *PredictTipSingleMode* demonstrates this approach. It takes as input multiple parameters representing feature values (for example, passenger count and trip distance), scores these features using the stored R model, and outputs the tip probability.

1. Run the following Transact-SQL statement to create the stored procedure.

SQL

```
USE [NYCTaxi_Sample]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P' AND name =
'PredictTipSingleMode')
DROP PROCEDURE v
GO

CREATE PROCEDURE [dbo].[PredictTipSingleMode] @passenger_count int = 0,
@trip_distance float = 0,
@trip_time_in_secs int = 0,
@pickup_latitude float = 0,
@pickup_longitude float = 0,
@dropoff_latitude float = 0,
@dropoff_longitude float = 0
AS
BEGIN
    DECLARE @inquiry nvarchar(max) = N'
        SELECT * FROM [dbo].[fnEngineerFeatures](@passenger_count,
@trip_distance, @trip_time_in_secs, @pickup_latitude,
```

```

@pickup_longitude, @dropoff_latitude, @dropoff_longitude)'
  DECLARE @lmodel2 varbinary(max) = (SELECT TOP 1 model FROM
nyc_taxi_models);

EXEC sp_execute_external_script @language = N'R', @script = N'
  mod <- unserialize(as.raw(model));
  print(summary(mod))
  OutputDataSet<-rxPredict(
    modelObject = mod,
    data = InputDataSet,
    outData = NULL,
    predVarNames = "Score",
    type = "response",
    writeModelVars = FALSE,
    overwrite = TRUE);
  str(OutputDataSet)
  print(OutputDataSet)
  ',
@input_data_1 = @inquery,
@params = N'
-- passthrough columns
@model varbinary(max) ,
@passenger_count int ,
@trip_distance float ,
@trip_time_in_secs int ,
@pickup_latitude float ,
@pickup_longitude float ,
@dropoff_latitude float ,
@dropoff_longitude float',
-- mapped variables
@model = @lmodel2 ,
@passenger_count =@passenger_count ,
@trip_distance=@trip_distance ,
@trip_time_in_secs=@trip_time_in_secs ,
@pickup_latitude=@pickup_latitude ,
@pickup_longitude=@pickup_longitude ,
@dropoff_latitude=@dropoff_latitude ,
@dropoff_longitude=@dropoff_longitude
WITH RESULT SETS ((Score float));
END

```

2. In SQL Server Management Studio, you can use the Transact-SQL EXEC procedure (or EXECUTE) to call the stored procedure, and pass it the required inputs. For example, try running this statement in Management Studio:

SQL

```
EXEC [dbo].[PredictTipSingleMode] 1, 2.5, 631, 40.763958,-73.973373,
40.782139,-73.977303
```

The values passed in here are, respectively, for the variables *passenger_count*, *trip_distance*, *trip_time_in_secs*, *pickup_latitude*, *pickup_longitude*, *dropoff_latitude*, and *dropoff_longitude*.

3. To run this same call from R code, you simply define an R variable that contains the entire stored procedure call, like this one:

R

```
q2 = "EXEC PredictTipSingleMode 1, 2.5, 631, 40.763958, -73.973373, 40.782139, -73.977303 ";
```

The values passed in here are, respectively, for the variables *passenger_count*, *trip_distance*, *trip_time_in_secs*, *pickup_latitude*, *pickup_longitude*, *dropoff_latitude*, and *dropoff_longitude*.

4. Call `sqlQuery` (from the **RODBC** package) and pass the connection string, together with the string variable containing the stored procedure call.

R

```
# predict with stored procedure in single mode  
sqlQuery (conn, q2);
```

Tip

R Tools for Visual Studio (RTVS) provides great integration with both SQL Server and R. See this article for more examples of using R ODBC with a SQL Server connection: [Working with SQL Server and R](#)

Next steps

Now that you have learned how to work with SQL Server data and persist trained R models to SQL Server, it should be relatively easy for you to create new models based on this data set. For example, you might try creating these additional models:

- A regression model that predicts the tip amount
- A multiclass classification model that predicts whether the tip is big, medium, or small

You might also want to explore these additional samples and resources:

- Data science scenarios and solution templates
- In-database advanced analytics

Tutorial: Use RevoScaleR R functions with SQL Server data

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

In this multi-part tutorial series, you're introduced to a range of **RevoScaleR** functions for tasks associated with data science. In the process, you'll learn how to create a remote compute context, move data between local and remote compute contexts, and execute R code on a remote SQL Server. You'll also learn how to analyze and plot data both locally and on the remote server, and how to create and deploy models.

RevoScaleR is a Microsoft R package providing distributed and parallel processing for data science and machine learning workloads. For R development in SQL Server, **RevoScaleR** is one of the core built-in packages, with functions for creating data source objects, setting a compute context, managing packages, and most importantly: working with data end-to-end, from import to visualization and analysis. Machine Learning algorithms in SQL Server have a dependency on **RevoScaleR** data sources. Given the importance of **RevoScaleR**, knowing when and how to call its functions is an essential skill.

Prerequisites

- [SQL Server Machine Learning Services](#) with the R feature, or [SQL Server R Services \(in-Database\)](#)
- [Database permissions](#) and a SQL Server database user login
- [SQL Server Management Studio](#)
- An IDE such as RStudio or the built-in RGUI tool included with R

To switch back and forth between local and remote compute contexts, you need two systems. Local is typically a development workstation with sufficient power for data science workloads. Remote in this case is SQL Server with the R feature enabled.

Switching compute contexts is predicated on having the same-version **RevoScaleR** on both local and remote systems. On a local workstation, you can get the **RevoScaleR** packages and related providers by installing Microsoft R Client.

If you need to put client and server on the same computer, be sure to install a second set of Microsoft R libraries for sending R script from a "remote" client. Do not use the R libraries that are installed in the program files of the SQL Server instance. Specifically, if you are using one computer, you need the **RevoScaleR** library in both of these locations to support client and server operations.

- C:\Program Files\Microsoft\R Client\R_SERVER\library\RevoScaleR
- C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library\RevoScaleR

For instructions on client configuration, see [Set up a data science client for R development](#).

R development tools

R developers typically use IDEs for writing and debugging R code. Here are some suggestions:

- **R Tools for Visual Studio** (RTVS) is a free plug-in that provides Intellisense, debugging, and support for Microsoft R. You can use it with SQL Server Machine Learning Services. To download, see [R Tools for Visual Studio](#).
- **RStudio** is one of the more popular environments for R development. For more information, see <https://www.rstudio.com/products/RStudio/>.
- Basic R tools (R.exe, RTerm.exe, RScripts.exe) are also installed by default when you install R in SQL Server or R Client. If you do not wish to install an IDE, you can use built-in R tools to execute the code in this tutorial.

Recall that **RevoScaleR** is required on both local and remote computers. You cannot complete this tutorial using a generic installation of RStudio or other environment that's missing the Microsoft R libraries. For more information, see [Set Up a Data Science Client](#).

Summary of tasks

- Data is initially obtained from CSV files or XDF files. You import the data into SQL Server using the functions in the **RevoScaleR** package.
- Model training and scoring is performed using the SQL Server compute context.
- Use **RevoScaleR** functions to create new SQL Server tables to save your scoring results.
- Create plots both on the server and in the local compute context.

- Train a model on data in SQL Server database, running R in the SQL Server instance.
- Extract a subset of data and save it as an XDF file for re-use in analysis on your local workstation.
- Get new data for scoring, by opening an ODBC connection to the SQL Server database. Scoring is done on the local workstation.
- Create a custom R function and run it in the server compute context to perform a simulation.

Next steps

[Tutorial 1: Create database and permissions](#)

Create a database and permissions (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 1 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

This tutorial describes how to create a SQL Server database and set the permissions necessary for completing the other tutorials in this series. Use [SQL Server Management Studio](#) or another query editor to complete the following tasks:

- ✓ Create a new database to store the data for training and scoring two R models
- ✓ Create a database user login with permissions for creating and using database objects

Create the database

This tutorial requires a database for storing data and code. If you're not an administrator, ask your DBA to create the database and login for you. You'll need permissions to write and read data, and to run R scripts.

1. In SQL Server Management Studio, connect to an R-enabled database instance.
2. Right-click **Databases**, and select **New database**.
3. Type a name for the new database: RevoDeepDive.

Create a login

1. Click **New Query**, and change the database context to the master database.
2. In the new **Query** window, run the following commands to create the user accounts and assign them to the database used for this tutorial. Be sure to change the database name if needed.
3. To verify the login, select the new database, expand **Security**, and expand **Users**.

Windows user

```

-- Create server user based on Windows account
USE master
GO
CREATE LOGIN [<DOMAIN>\<user_name>] FROM WINDOWS WITH DEFAULT_DATABASE=
[RevoDeepDive]

--Add the new user to tutorial database
USE [RevoDeepDive]
GO
CREATE USER [<user_name>] FOR LOGIN [<DOMAIN>\<user_name>] WITH
DEFAULT_SCHEMA=[db_datareader]

```

SQL login

SQL

```

-- Create new SQL login
USE master
GO
CREATE LOGIN [DDUser01] WITH PASSWORD='<type password here>',
CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF;

-- Add the new SQL login to tutorial database
USE RevoDeepDive
GO
CREATE USER [DDUser01] FOR LOGIN [DDUser01] WITH DEFAULT_SCHEMA=
[db_datareader]

```

Assign permissions

This tutorial demonstrates R script and DDL operations, including creating and deleting tables and stored procedures, and running R script in an external process on SQL Server. In this step, assign permissions to allow these tasks.

This example assumes a SQL login (DDUser01), but if you created a Windows login, use that instead.

SQL

```

USE RevoDeepDive
GO

EXEC sp_addrolemember 'db_owner', 'DDUser01'
GRANT EXECUTE ANY EXTERNAL SCRIPT TO DDUser01
GO

```

Troubleshoot connections

This section lists some common issues that you might run across in the course of setting up the database.

- **How can I verify database connectivity and check SQL queries?**

Before you run R code using the server, you might want to check that the database can be reached from your R development environment. Both [Server Explorer in Visual Studio](#) and [SQL Server Management Studio](#) are free tools with powerful database connectivity and management features.

If you don't want to install additional database management tools, you can create a test connection to the SQL Server instance by using the [ODBC Data Source Administrator](#) in Control Panel. If the database is configured correctly and you enter the correct user name and password, you should be able to see the database you just created and select it as your default database.

Common reasons for connection failures include remote connections are not enabled for the server, and Named Pipes protocol is not enabled. You can find more troubleshooting tips in this article: [Troubleshoot Connecting to the SQL Server Database Engine](#).

- **My table name has datareader prefixed to it - why?**

When you specify the default schema for this user as **db_datareader**, all tables and other new objects created by this user are prefixed with the *schema* name. A schema is like a folder that you can add to a database to organize objects. The schema also defines a user's privileges within the database.

When the schema is associated with one particular user name, the user is the *schema owner*. When you create an object, you always create it in your own schema, unless you specifically ask it to be created in another schema.

For example, if you create a table with the name **TestData**, and your default schema is **db_datareader**, the table is created with the name

```
<database_name>.db_datareader.TestData.
```

For this reason, a database can contain multiple tables with the same names, as long as the tables belong to different schemas.

If you are looking for a table and do not specify a schema, the database server looks for a schema that you own. Therefore, there is no need to specify the schema name when accessing tables in a schema associated with your login.

- I don't have DDL privileges. Can I still run the tutorial??

Yes, but you should ask someone to pre-load the data into the SQL Server tables, and skip ahead to the next tutorial. The functions that require DDL privileges are called out in the tutorial wherever possible.

Also, ask your administrator to grant you the permission, EXECUTE ANY EXTERNAL SCRIPT. It is needed for R script execution, whether remote or by using `sp_execute_external_script`.

Next steps

Create SQL Server data objects using `RxSqlServerData`

Create SQL Server data objects using RxSqlServerData (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 2 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

This tutorial is a continuation of database creation: adding tables and loading data. If a DBA created the database and login in [tutorial two](#), you can add tables using an R IDE like RStudio or a built-in tool like **Rgui**.

From R, connect to SQL Server and use **RevoScaleR** functions to perform the following tasks:

- ✓ Create tables for training data and predictions
- ✓ Load tables with data from a local .csv file

Sample data is simulated credit card fraud data (the ccFraud dataset), partitioned into training and scoring datasets. The data file is included in **RevoScaleR**.

Use an R IDE or **Rgui** to complete these tasks. Be sure to use the R executables found at this location: C:\Program Files\Microsoft\R Client\R_SERVER\bin\x64 (either Rgui.exe if you are using that tool, or an R IDE pointing to C:\Program Files\Microsoft\R Client\R_SERVER). Having an [R client workstation](#) with these executables is considered a prerequisite of this tutorial.

Create the training data table

1. Store the database connection string in an R variable. Below are two examples of valid ODBC connection strings for SQL Server: one using a SQL login, and one for Windows integrated authentication.

Be sure to modify the server name, user name, and password as appropriate.

SQL login

```
R
```

```
sqlConnString <- "Driver=SQL Server;Server=<server-name>;  
Database=RevoDeepDive;Uid=<user_name>;Pwd=<password>"
```

Windows authentication

R

```
sqlConnString <- "Driver=SQL Server;Server=<server-  
name>;Database=RevoDeepDive;Trusted_Connection=True"
```

2. Specify the name of the table you want to create, and save it in an R variable.

R

```
sqlFraudTable <- "ccFraudSmall"
```

Because the server instance and database name are already specified as part of the connection string, when you combine the two variables, the *fully qualified* name of the new table becomes *instance.database.schema.ccFraudSmall*.

3. Optionally, specify *rowsPerRead* to control how many rows of data are read in each batch.

R

```
sqlRowsPerRead = 5000
```

Although this parameter is optional, setting it can result in more efficient computations. Most of the enhanced analytical functions in **RevoScaleR** and **MicrosoftML** process data in chunks. The *rowsPerRead* parameter determines the number of rows in each chunk.

You might need to experiment with this setting to find the right balance. If the value is too large, data access might be slow if there is not enough memory to process data in chunks of that size. Conversely, on some systems, if the value of *rowsPerRead* is too small, performance can also slow down.

As an initial value, use the default batch process size defined by the database engine instance to control the number of rows in each chunk (5,000 rows). Save that value in the variable *sqlRowsPerRead*.

4. Define a variable for the new data source object, and pass the arguments previously defined to the **RxSqlServerData** constructor. Note that this only creates

the data source object and does not populate it. Loading data is a separate step.

R

```
sqlFraudDS <- RxSqlServerData(connectionString = sqlConnString,  
  table = sqlFraudTable,  
  rowsPerRead = sqlRowsPerRead)
```

Create the scoring data table

Using the same steps, create the table that holds the scoring data using the same process.

1. Create a new R variable, *sqlScoreTable*, to store the name of the table used for scoring.

R

```
sqlScoreTable <- "ccFraudScoreSmall"
```

2. Provide that variable as an argument to the **RxSqlServerData** function to define a second data source object, *sqlScoreDS*.

R

```
sqlScoreDS <- RxSqlServerData(connectionString = sqlConnString,  
  table = sqlScoreTable, rowsPerRead = sqlRowsPerRead)
```

Because you've already defined the connection string and other parameters as variables in the R workspace, you can reuse it for new data sources representing different tables, views, or queries.

ⓘ Note

The function uses different arguments for defining a data source based on an entire table than for a data source based on a query. This is because the SQL Server database engine must prepare the queries differently. Later in this tutorial, you learn how to create a data source object based on a SQL query.

Load data into SQL tables using R

Now that you have created the SQL Server tables, you can load data into them using the appropriate **Rx** function.

The **RevoScaleR** package contains functions specific to data source types. For text data, use **RxTextData** to generate the data source object. There are additional functions for creating data source objects from Hadoop data, ODBC data, and so forth.

ⓘ Note

For this section, you must have **Execute DDL** permissions on the database.

Load data into the training table

1. Create an R variable, *ccFraudCsv*, and assign to the variable the file path for the CSV file containing the sample data. This dataset is provided in **RevoScaleR**. The "sampleDataDir" is a keyword on the **rxGetOption** function.

R

```
ccFraudCsv <- file.path(rxGetOption("sampleDataDir"),  
"ccFraudSmall.csv")
```

Notice the call to **rxGetOption**, which is the GET method associated with **rxOptions** in **RevoScaleR**. Use this utility to set and list options related to local and remote compute contexts, such as the default shared directory, or the number of processors (cores) to use in computations.

This particular call gets the samples from the correct library, regardless of where you are running your code. For example, try running the function on SQL Server, and on your development computer, and see how the paths differ.

2. Define a variable to store the new data, and use the **RxTextData** function to specify the text data source.

R

```
inTextData <- RxTextData(file = ccFraudCsv, colClasses = c(  
  "custID" = "integer", "gender" = "integer", "state" = "integer",  
  "cardholder" = "integer", "balance" = "integer",  
  "numTrans" = "integer",  
  "numIntlTrans" = "integer", "creditLine" = "integer",  
  "fraudRisk" = "integer"))
```

The argument `colClasses` is important. You use it to indicate the data type to assign to each column of data loaded from the text file. In this example, all columns are handled as text, except for the named columns, which are handled as integers.

3. At this point, you might want to pause a moment, and view your database in SQL Server Management Studio. Refresh the list of tables in the database.

You can see that, although the R data objects have been created in your local workspace, the tables have not been created in the SQL Server database. Also, no data has been loaded from the text file into the R variable.

4. Insert the data by calling the function `rxDataStep` function.

```
R
rxDataStep(inData = inTextData, outFile = sqlFraudDS, overwrite = TRUE)
```

Assuming no problems with your connection string, after a brief pause, you should see results like these:

Total Rows written: 10000, Total time: 0.466 Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.577 seconds

5. Refresh the list of tables. To verify that each variable has the correct data types and was imported successfully, you can also right-click the table in SQL Server Management Studio and select **Select Top 1000 Rows**.

Load data into the scoring table

1. Repeat the steps to load the data set used for scoring into the database.

Start by providing the path to the source file.

```
R
ccScoreCsv <- file.path(rxGetOption("sampleDataDir"),
"ccFraudScoreSmall.csv")
```

2. Use the `RxTextData` function to get the data and save it in the variable, `inTextData`.

```
R
inTextData <- RxTextData(file = ccScoreCsv, colClasses = c(
"custID" = "integer", "gender" = "integer", "state" = "integer",
"cardholder" = "integer", "balance" = "integer",
```

```
"numTrans" = "integer",  
"numIntlTrans" = "integer", "creditLine" = "integer"))
```

3. Call the `rxDataStep` function to overwrite the current table with the new schema and data.

R

```
rxDataStep(inData = inTextData, sqlScoreDS, overwrite = TRUE)
```

- The *inData* argument defines the data source to use.
- The *outFile* argument specifies the table in SQL Server where you want to save the data.
- If the table already exists and you don't use the *overwrite* option, results are inserted without truncation.

Again, if the connection was successful, you should see a message indicating completion and the time required to write the data into the table:

```
Total Rows written: 10000, Total time: 0.384 Rows Read: 10000, Total Rows Processed:  
10000, Total Chunk Time: 0.456 seconds
```

More about rxDataStep

`rxDataStep` is a powerful function that can perform multiple transformations on an R data frame. You can also use `rxDataStep` to convert data into the representation required by the destination: in this case, SQL Server.

Optionally, you can specify transformations on the data, by using R functions in the arguments to `rxDataStep`. Examples of these operations are provided later in this tutorial.

Next steps

Query and modify the SQL Server data

Query and modify the SQL Server data (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 3 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

In the previous tutorial, you loaded the data into SQL Server. In this tutorial, you can explore and modify data using **RevoScaleR**:

- ✓ Return basic information about the variables
- ✓ Create categorical data from raw data

Categorical data, or *factor variables*, are useful for exploratory data visualizations. You can use them as inputs to histograms to get an idea of what variable data looks like.

Query for columns and types

Use an R IDE or RGui.exe to run R script.

First, get a list of the columns and their data types. You can use the function [rxGetVarInfo](#) and specify the data source you want to analyze. Depending on your version of **RevoScaleR**, you could also use [rxGetVarNames](#).

```
R
```

```
rxGetVarInfo(data = sqlFraudDS)
```

Results

```
R
```

```
Var 1: custID, Type: integer  
Var 2: gender, Type: integer  
Var 3: state, Type: integer  
Var 4: cardholder, Type: integer  
Var 5: balance, Type: integer  
Var 6: numTrans, Type: integer  
Var 7: numIntlTrans, Type: integer  
Var 8: creditLine, Type: integer  
Var 9: fraudRisk, Type: integer
```

Create categorical data

All the variables are stored as integers, but some variables represent categorical data, called *factor variables* in R. For example, the column *state* contains numbers used as identifiers for the 50 states plus the District of Columbia. To make it easier to understand the data, you replace the numbers with a list of state abbreviations.

In this step, you create a string vector containing the abbreviations, and then map these categorical values to the original integer identifiers. Then you use the new variable in the *colInfo* argument, to specify that this column be handled as a factor. Whenever you analyze the data or move it, the abbreviations are used and the column is handled as a factor.

Mapping the column to abbreviations before using it as a factor actually improves performance as well. For more information, see [R and data optimization](#).

1. Begin by creating an R variable, *stateAbb*, and defining the vector of strings to add to it, as follows.

```
R

stateAbb <- c("AK", "AL", "AR", "AZ", "CA", "CO", "CT", "DC",
             "DE", "FL", "GA", "HI", "IA", "ID", "IL", "IN", "KS", "KY", "LA",
             "MA", "MD", "ME", "MI", "MN", "MO", "MS", "MT", "NB", "NC", "ND",
             "NH", "NJ", "NM", "NV", "NY", "OH", "OK", "OR", "PA", "RI", "SC",
             "SD", "TN", "TX", "UT", "VA", "VT", "WA", "WI", "WV", "WY")
```

2. Next, create a column information object, named *ccColInfo*, that specifies the mapping of the existing integer values to the categorical levels (the abbreviations for states).

This statement also creates factor variables for gender and cardholder.

```
R

ccColInfo <- list(
  gender = list(
    type = "factor",
    levels = c("1", "2"),
    newLevels = c("Male", "Female")
  ),
  cardholder = list(
    type = "factor",
    levels = c("1", "2"),
    newLevels = c("Principal", "Secondary")
  ),
  state = list(
```

```
    type = "factor",
    levels = as.character(1:51),
    newLevels = stateAbb
  ),
  balance = list(type = "numeric")
)
```

3. To create the SQL Server data source that uses the updated data, call the **RxSqlServerData** function as before, but add the *colInfo* argument.

R

```
sqlFraudDS <- RxSqlServerData(connectionString = sqlConnString,
  table = sqlFraudTable, colInfo = ccColInfo,
  rowsPerRead = sqlRowsPerRead)
```

- For the *table* parameter, pass in the variable *sqlFraudTable*, which contains the data source you created earlier.
- For the *colInfo* parameter, pass in the *ccColInfo* variable, which contains the column data types and factor levels.

4. You can now use the function **rxGetVarInfo** to view the variables in the new data source.

R

```
rxGetVarInfo(data = sqlFraudDS)
```

Results

R

```
Var 1: custID, Type: integer
Var 2: gender  2 factor levels: Male Female
Var 3: state   51 factor levels: AK AL AR AZ CA ... VT WA WI WV WY
Var 4: cardholder  2 factor levels: Principal Secondary
Var 5: balance, Type: integer
Var 6: numTrans, Type: integer
Var 7: numIntlTrans, Type: integer
Var 8: creditLine, Type: integer
Var 9: fraudRisk, Type: integer
```

Now the three variables you specified (*gender*, *state*, and *cardholder*) are treated as factors.

Next steps

Define and use compute contexts

Define and use compute contexts (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 4 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

In the previous tutorial, you used **RevoScaleR** functions to inspect data objects. This tutorial introduces the [RxInSqlServer](#) function, which lets you define a compute context for a remote SQL Server. With a remote compute context, you can shift R execution from a local session to a remote session on the server.

- ✓ Learn the elements of a remote SQL Server compute context
- ✓ Enable tracing on a compute context object

RevoScaleR supports multiple compute contexts: Hadoop, Spark on HDFS, and SQL Server in-database. For SQL Server, the **RxInSqlServer** function is used for server connections and passing objects between the local computer and the remote execution context.

Create and set a compute context

The **RxInSqlServer** function that creates the SQL Server compute context uses the following information:

- Connection string for the SQL Server instance
- Specification of how output should be handled
- Optional specification of a shared data directory
- Optional arguments that enable tracing or specify the trace level

This section walks you through each part.

1. Specify the connection string for the instance where computations are performed. You can re-use the connection string that you created earlier.

Using a SQL login

R


```
sqlConnString <- "Driver=SQL Server;Server=<SQL Server instance name>;  
Database=<database name>;Uid=<SQL user nme>;Pwd=<password>"
```

Using Windows authentication

R

```
sqlConnString <- "Driver=SQL  
Server;Server=instance_name;Database=RevoDeepDive;Trusted_Connection=Tr  
ue"
```

2. Specify how you want the output handled. The following script directs the local R session to wait for R job results on the server before processing the next operation. It also suppresses output from remote computations from appearing in the local session.

R

```
sqlWait <- TRUE  
sqlConsoleOutput <- FALSE
```

The *wait* argument to `RxInSqlServer` supports these options:

- **TRUE.** The job is configured as blocking and does not return until it has completed or has failed.
- **FALSE.** Jobs are configured as non-blocking and return immediately, allowing you to continue running other R code. However, even in non-blocking mode, the client connection with SQL Server must be maintained while the job is running.

3. Optionally, specify the location of a local directory for shared use by the local R session and by the remote SQL Server computer and its accounts.

R

```
sqlShareDir <- paste("c:\\AllShare\\", Sys.getenv("USERNAME"), sep="")
```

If you want to manually create a specific directory for sharing, you can add a line like the following:

R

```
dir.create(sqlShareDir, recursive = TRUE)
```

4. Pass arguments to the **RxInSqlServer** constructor to create the *compute context object*.

```
R
```

```
sqlCompute <- RxInSqlServer(  
  connectionString = sqlConnString,  
  wait = sqlWait,  
  consoleOutput = sqlConsoleOutput)
```

The syntax for **RxInSqlServer** looks almost identical to that of the **RxSqlServerData** function that you used earlier to define the data source. However, there are some important differences.

- The data source object, defined by using the function [RxSqlServerData](#), specifies where the data is stored.
- In contrast, the compute context, defined by using the function [RxInSqlServer](#) indicates where aggregations and other computations are to take place.

Defining a compute context does not affect any other generic R computations that you might perform on your workstation, and does not change the source of the data. For example, you could define a local text file as the data source but change the compute context to SQL Server and do all your reading and summaries on the data on the SQL Server computer.

5. Activate the remote compute context.

```
R
```

```
rxSetComputeContext(sqlCompute)
```

6. Return information about the compute context, including its properties.

```
R
```

```
rxGetComputeContext()
```

7. Reset the compute context back to the local computer by specifying the "local" keyword (the next tutorial demonstrates using the remote compute context).

```
R
```

```
rxSetComputeContext("local")
```

💡 Tip

For a list of other keywords supported by this function, type `help("rxSetComputeContext")` from an R command line.

Enable tracing

Sometimes operations work on your local context, but have issues when running in a remote compute context. If you want to analyze issues or monitor performance, you can enable tracing in the compute context, to support run-time troubleshooting.

1. Create a new compute context that uses the same connection string, but add the arguments *traceEnabled* and *traceLevel* to the **RxInSqlServer** constructor.

```
R

sqlComputeTrace <- RxInSqlServer(
  connectionString = sqlConnString,
  #shareDir = sqlShareDir,
  wait = sqlWait,
  consoleOutput = sqlConsoleOutput,
  traceEnabled = TRUE,
  traceLevel = 7)
```

In this example, the *traceLevel* property is set to 7, meaning "show all tracing information."

2. Use the [rxSetComputeContext](#) function to specify the tracing-enabled compute context by name.

```
R

rxSetComputeContext(sqlComputeTrace)
```

Next steps

Learn how to switch compute contexts to run R code on the server or locally.

[Compute summary statistics in local and remote compute contexts](#)

Compute summary statistics in R (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 5 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

This tutorial uses the established data sources and compute contexts created in previous tutorials to run high-powered R scripts. In this tutorial, you will use local and remote server compute contexts for the following tasks:

- ✓ Switch the compute context to SQL Server
- ✓ Obtain summary statistics on remote data objects
- ✓ Compute a local summary

If you completed the previous tutorials, you should have these remote compute contexts: `sqlCompute` and `sqlComputeTrace`. Moving forward, you use will `sqlCompute` and the local compute context in subsequent tutorials.

Use an R IDE or **Rgui** to run the R script in this tutorial.

Compute summary statistics on remote data

Before you can run any R code remotely, you need to specify the remote compute context. All subsequent computations take place on the SQL Server computer specified in the `sqlCompute` parameter.

A compute context remains active until you change it. However, any R scripts that *cannot* run in a remote server context will automatically run locally.

To see how a compute context works, generate summary statistics on the `sqlFraudDS` data source on the remote SQL Server. This data source object was created in [tutorial two](#) and represents the `ccFraudSmall` table in the `RevoDeepDive` database.

1. Switch the compute context to `sqlCompute` created in the previous tutorial:

```
R  
  
rxSetComputeContext(sqlCompute)
```

2. Call the `rxSummary` function and pass required arguments, such as the formula and the data source, and assign the results to the variable `sumOut`.

R

```
sumOut <- rxSummary(formula = ~gender + balance + numTrans +  
numIntlTrans + creditLine, data = sqlFraudDS)
```

The R language provides many summary functions, but `rxSummary` in **RevoScaleR** supports execution on various remote compute contexts, including SQL Server. For information about similar functions, see [Data summaries using RevoScaleR](#).

3. Print the contents of `sumOut` to the console.

R

```
sumOut
```

ⓘ Note

If you get an error, wait a few minutes for execution to finish before retrying the command.

Results

R

```
Summary Statistics Results for: ~gender + balance + numTrans + numIntlTrans  
+ creditLine
```

```
Data: sqlFraudDS (RxSqlServerData Data Source)
```

```
Number of valid observations: 10000
```

Name	Mean	StdDev	Min	Max	ValidObs	MissingObs		
balance	4075.0318	3926.558714			0	25626	100000	
numTrans	29.1061	26.619923	0		100	10000	0	100000
numIntlTrans	4.0868	8.726757	0		60	10000	0	100000
creditLine	9.1856	9.870364	1		75	10000	0	100000

```
Category Counts for gender
```

```
Number of categories: 2
```

```
Number of valid observations: 10000
```

```
Number of missing observations: 0
```

```
gender Counts
```

```
Male 6154
```

```
Female 3846
```

Create a local summary

1. Change the compute context to do all your work locally.

```
R
```

```
rxSetComputeContext ("local")
```

2. When extracting data from SQL Server, you can often get better performance by increasing the number of rows extracted for each read, assuming the increased block size can be accommodated in memory. Run the following command to increase the value for the *rowsPerRead* parameter on the data source. Previously, the value of *rowsPerRead* was set to 5000.

```
R
```

```
sqlServerDS1 <- RxSqlServerData(  
  connectionString = sqlConnString,  
  table = sqlFraudTable,  
  colInfo = ccColInfo,  
  rowsPerRead = 10000)
```

3. Call **rxSummary** on the new data source.

```
R
```

```
rxSummary(formula = ~gender + balance + numTrans + numIntlTrans +  
  creditLine, data = sqlServerDS1)
```

The actual results should be the same as when you run **rxSummary** in the context of the SQL Server computer. However, the operation might be faster or slower. Much depends on the connection to your database, because the data is being transferred to your local computer for analysis.

4. Switch back to the remote compute context for the next several tutorials.

```
R
```

```
rxSetComputeContext(sqlCompute)
```

Next steps

Visualize SQL Server data using R

Visualize SQL Server data using R (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 6 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

In this tutorial, you'll use R functions to view the distribution of values in the *creditLine* column by gender.

- ✓ Create min-max variables for histogram inputs
- ✓ Visualize data in a histogram using `rxHistogram` from **RevoScaleR**
- ✓ Visualize with scatter plots using `levelplot` from **lattice** included in the base R distribution

As this tutorial demonstrates, you can combine open-source and Microsoft-specific functions in the same script.

Add maximum and minimum values

Based on the computed summary statistics from the previous tutorial, you've discovered some useful information about the data that you can insert into the data source for further computations. For example, the minimum and maximum values can be used to compute histograms. In this exercise, add the high and low values to the `RxSqlServerData` data source.

1. Start by setting up some temporary variables.

```
R  
  
sumDF <- sumOut$sDataFrame  
var <- sumDF$Name
```

2. Use the variable `ccCollInfo` that you created in the previous tutorial to define the columns in the data source.

Add new computed columns (`numTrans`, `numIntlTrans`, and `creditLine`) to the column collection that override the original definition. The script below adds

factors based on minimum and maximum values, obtained from `sumOut`, which is storing the in-memory output from `rxSummary`.

```
R

ccColInfo <- list(
  gender = list(type = "factor",
    levels = c("1", "2"),
    newLevels = c("Male", "Female")),
  cardholder = list(type = "factor",
    levels = c("1", "2"),
    newLevels = c("Principal", "Secondary")),
  state = list(type = "factor",
    levels = as.character(1:51),
    newLevels = stateAbb),
  balance = list(type = "numeric"),
  numTrans = list(type = "factor",
    levels = as.character(sumDF[var == "numTrans", "Min"]:sumDF[var
== "numTrans", "Max"])),
  numIntlTrans = list(type = "factor",
    levels = as.character(sumDF[var == "numIntlTrans",
"Min"]:sumDF[var == "numIntlTrans", "Max"])),
  creditLine = list(type = "numeric")
)
```

3. Having updated the column collection, apply the following statement to create an updated version of the SQL Server data source that you defined earlier.

```
R

sqlFraudDS <- RxSqlServerData(
  connectionString = sqlConnString,
  table = sqlFraudTable,
  colInfo = ccColInfo,
  rowsPerRead = sqlRowsPerRead)
```

The `sqlFraudDS` data source now includes the new columns added using `ccColInfo`.

At this point, the modifications affect only the data source object in R; no new data has been written to the database table yet. However, you can use the data captured in the `sumOut` variable to create visualizations and summaries.

Tip

If you forget which compute context you're using, run `rxGetComputeContext()`. A return value of "RxLocalSeq Compute Context" indicates that you are running in the local compute context.

Visualize data using rxHistogram

1. Use the following R code to call the `rxHistogram` function and pass a formula and data source. You can run this locally at first, to see the expected results, and how long it takes.

R

```
rxHistogram(~creditLine|gender, data = sqlFraudDS, histType =  
"Percent")
```

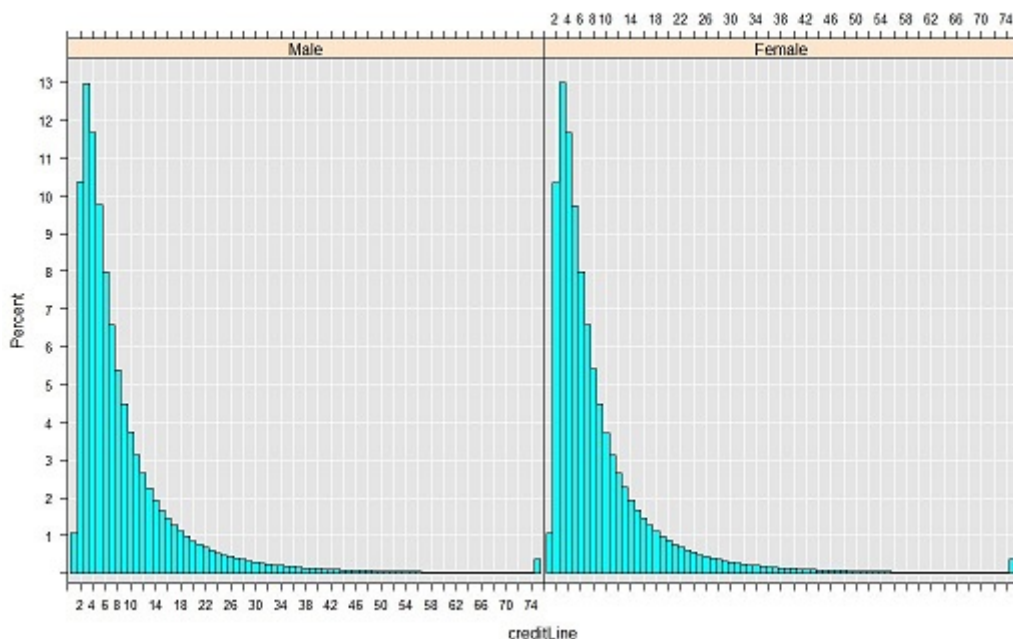
Internally, `rxHistogram` calls the `rxCube` function, which is included in the **RevoScaleR** package. `rxCube` outputs a single list (or data frame) containing one column for each variable specified in the formula, plus a counts column.

2. Now, set the compute context to the remote SQL Server computer and run `rxHistogram` again.

R

```
rxSetComputeContext(sqlCompute)  
rxHistogram(~creditLine|gender, data = sqlFraudDS, histType =  
"Percent")
```

3. The results are exactly the same because you're using the same data source, but in the second step, the computations are performed on the remote server. The results are then returned to your local workstation for plotting.



Visualize with scatter plots

Scatter plots are often used during data exploration to compare the relationship between two variables. You can use built-in R packages for this purpose, with inputs provided by **RevoScaleR** functions.

1. Call the **rxCube** function to compute the mean of *fraudRisk* for every combination of *numTrans* and *numInt1Trans*:

R

```
cube1 <- rxCube(fraudRisk~F(numTrans):F(numInt1Trans), data =  
sqlFraudDS)
```

To specify the groups used to compute group means, use the **F()** notation. In this example, **F(numTrans):F(numInt1Trans)** indicates that the integers in the variables **numTrans** and **numInt1Trans** should be treated as categorical variables, with a level for each integer value.

The default return value of **rxCube** is an *rxCube object*, which represents a cross-tabulation.

2. Call **rxResultsDF** function to convert the results into a data frame that can easily be used in one of R's standard plotting functions.

R

```
cubePlot <- rxResultsDF(cube1)
```

The **rxCube** function includes an optional argument, *returnDataFrame* = **TRUE**, that you could use to convert the results to a data frame directly. For example:

```
print(rxCube(fraudRisk~F(numTrans):F(numInt1Trans), data = sqlFraudDS,  
returnDataFrame = TRUE))
```

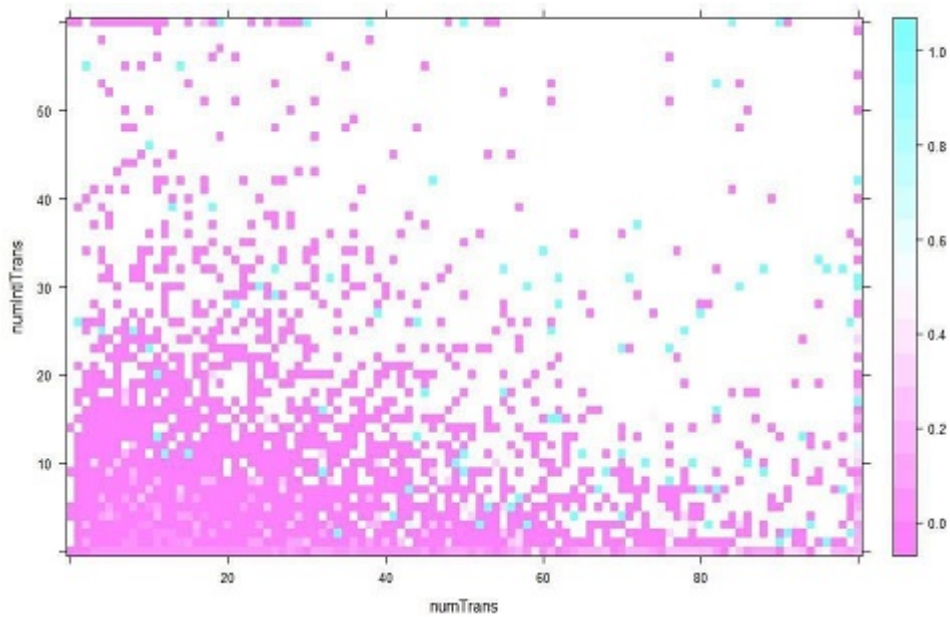
However, the output of **rxResultsDF** is cleaner and preserves the names of the source columns. You can run **head(cube1)** followed by **head(cubePlot)** to compare the output.

3. Create a heat map using the **levelplot** function from the **lattice** package, included with all R distributions.

R

```
levelplot(fraudRisk~numTrans*numIntlTrans, data = cubePlot)
```

Results



From this quick analysis, you can see that the risk of fraud increases with both the number of transactions and the number of international transactions.

For more information about the `rxCube` function and crosstabs in general, see [Data summaries using RevoScaleR](#).

Next steps

Create R models using SQL Server data

Create R models (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 7 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

You have enriched the training data. In this tutorial, you'll analyze the data using regression modeling. Linear models are an important tool in the world of predictive analytics. The **RevoScaleR** package includes regression algorithms that can subdivide the workload and run it in parallel.

- ✓ Create a linear regression model
- ✓ Create a logistic regression model

Create a linear regression model

In this step, create a simple linear model that estimates the credit card balance for the customers using as independent variables the values in the *gender* and *creditLine* columns.

To do this, use the [rxLinMod](#) function, which supports remote compute contexts.

1. Create an R variable to store the completed model, and call **rxLinMod**, passing an appropriate formula.

R

```
linModObj <- rxLinMod(balance ~ gender + creditLine, data = sqlFraudDS)
```

2. To view a summary of the results, call the standard R **summary** function on the model object.

R

```
summary(linModObj)
```

You might think it peculiar that a plain R function like `summary` would work here, since in the previous step, you set the compute context to the server. However, even when the `rxLinMod` function uses the remote compute context to create the model, it also returns an object that contains the model to your local workstation, and stores it in the shared directory.

Therefore, you can run standard R commands against the model just as if it had been created using the "local" context.

Results

R

```
Linear Regression Results for: balance ~ gender + creditLineData: sqlFraudDS
(RxSqlServerData Data Source)
Dependent variable(s): balance
Total independent variables: 4 (Including number dropped: 1)
Number of valid observations: 10000
Number of missing observations: 0
Coefficients: (1 not defined because of singularities)

Estimate Std. Error t value Pr(>|t|) (Intercept)
3253.575 71.194 45.700 2.22e-16
gender=Male -88.813 78.360 -1.133 0.257
gender=Female Dropped Dropped Dropped Dropped
creditLine 95.379 3.862 24.694 2.22e-16
Signif. codes: 0 0.001 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3812 on 9997 degrees of freedom
Multiple R-squared: 0.05765
Adjusted R-squared: 0.05746
F-statistic: 305.8 on 2 and 9997 DF, p-value: < 2.2e-16
Condition number: 1.0184
```

Create a logistic regression model

Next, create a logistic regression model that indicates whether a particular customer is a fraud risk. You'll use the `RevoScaleR rxLogit` function, which supports fitting of logistic regression models in remote compute contexts.

Keep the compute context as is. You'll also continue to use the same data source as well.

1. Call the `rxLogit` function and pass the formula needed to define the model.

R

```
logitObj <- rxLogit(fraudRisk ~ state + gender + cardholder + balance +
  numTrans + numIntlTrans + creditLine, data = sqlFraudDS, dropFirst =
```

```
TRUE)
```

Because it is a large model, containing 60 independent variables, including three dummy variables that are dropped, you might have to wait some time for the compute context to return the object.

The reason the model is so large is that, in R and in the **RevoScaleR** package, every level of a categorical factor variable is automatically treated as a separate dummy variable.

2. To view a summary of the returned model, call the R **summary** function.

```
R
```

```
summary(logitObj)
```

Partial results

```
R
```

```
Logistic Regression Results for: fraudRisk ~ state + gender + cardholder +
balance + numTrans + numIntlTrans + creditLine
Data: sqlFraudDS (RxSqlServerData Data Source)
Dependent variable(s): fraudRisk
Total independent variables: 60 (Including number dropped: 3)
Number of valid observations: 10000 -2
```

```
LogLikelihood: 2032.8699 (Residual deviance on 9943 degrees of freedom)
```

```
Coefficients:
```

```
Estimate Std. Error z value Pr(>|z|)      (Intercept)
```

```
-8.627e+00  1.319e+00  -6.538  6.22e-11
```

```
state=AK           Dropped      Dropped Dropped  Dropped
```

```
state=AL           -1.043e+00  1.383e+00  -0.754   0.4511
```

```
(other states omitted)
```

```
gender=Male        Dropped      Dropped Dropped  Dropped
```

```
gender=Female      7.226e-01   1.217e-01   5.936   2.92e-09
```

```
cardholder=Principal Dropped      Dropped Dropped  Dropped
```

```
cardholder=Secondary 5.635e-01   3.403e-01   1.656   0.0977
```

```
balance            3.962e-04   1.564e-05   25.335   2.22e-16
```

```
numTrans           4.950e-02   2.202e-03   22.477   2.22e-16
```

```
numIntlTrans       3.414e-02   5.318e-03   6.420   1.36e-10
```

```
creditLine         1.042e-01   4.705e-03   22.153   2.22e-16
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Condition number of final variance-covariance matrix: 3997.308
```

```
Number of iterations: 15
```

Next steps

Score new data

Score new data (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 8 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

In this tutorial, you'll use the logistic regression model that you created in the previous tutorial to score another data set that uses the same independent variables as inputs.

- ✓ Score new data
- ✓ Create a histogram of the scores

Note

You need DDL admin privileges for some of these steps.

Generate and save scores

1. Update the sqlScoreDS data source (created in [tutorial two](#)) to use column information created in the previous tutorial.

R

```
sqlScoreDS <- RxSqlServerData(  
  connectionString = sqlConnString,  
  table = sqlScoreTable,  
  colInfo = ccColInfo,  
  rowsPerRead = sqlRowsPerRead)
```

2. To make sure you don't lose the results, create a new data source object. Then, use the new data source object to populate a new table in the RevoDeepDive database.

R

```
sqlServerOutDS <- RxSqlServerData(table = "ccScoreOutput",  
  connectionString = sqlConnString,  
  rowsPerRead = sqlRowsPerRead )
```


At this point, the table has not been created. This statement just defines a container for the data.

3. Check the current compute context using `rxGetComputeContext()`, and set the compute context to the server if needed.

```
R  
  
rxSetComputeContext(sqlCompute)
```

4. As a precaution, check for the existence of the output table. If one already exists with the same name, you will get an error when attempting to write the new table.

To do this, make a call to the functions `rxSqlServerTableExists` and `rxSqlServerDropTable`, passing the table name as input.

```
R  
  
if (rxSqlServerTableExists("ccScoreOutput"))  
  rxSqlServerDropTable("ccScoreOutput")
```

- `rxSqlServerTableExists` queries the ODBC driver and returns TRUE if the table exists, FALSE otherwise.
- `rxSqlServerDropTable` executes the DDL and returns TRUE if the table is successfully dropped, FALSE otherwise.

5. Execute `rxPredict` to create the scores, and save them in the new table defined in data source `sqlScoreDS`.

```
R  
  
rxPredict(modelObject = logitObj,  
          data = sqlScoreDS,  
          outData = sqlServerOutDS,  
          predVarNames = "ccFraudLogitScore",  
          type = "link",  
          writeModelVars = TRUE,  
          overwrite = TRUE)
```

The `rxPredict` function is another function that supports running in remote compute contexts. You can use the `rxPredict` function to create scores from models based on `rxLinMod`, `rxLogit`, or `rxGlm`.

- The parameter `writeModelVars` is set to `TRUE` here. This means that the variables that were used for estimation will be included in the new table.

- The parameter *predVarNames* specifies the variable where results will be stored. Here you are passing a new variable, `ccFraudLogitScore`.
 - The *type* parameter for `rxPredict` defines how you want the predictions calculated. Specify the keyword **response** to generate scores based on the scale of the response variable. Or, use the keyword **link** to generate scores based on the underlying link function, in which case predictions are created using a logistic scale.
6. After a while, you can refresh the list of tables in Management Studio to see the new table and its data.
 7. To add additional variables to the output predictions, use the *extraVarsToWrite* argument. For example, in the following code, the variable *custID* is added from the scoring data table into the output table of predictions.

```
R

rxPredict(modelObject = logitObj,
          data = sqlScoreDS,
          outData = sqlServerOutDS,
          predVarNames = "ccFraudLogitScore",
          type = "link",
          writeModelVars = TRUE,
          extraVarsToWrite = "custID",
          overwrite = TRUE)
```

Display scores in a histogram

After the new table has been created, compute and display a histogram of the 10,000 predicted scores. Computation is faster if you specify the low and high values, so get those from the database and add them to your working data.

1. Create a new data source, `sqlMinMax`, that queries the database to get the low and high values.

```
R

sqlMinMax <- RxSqlServerData(
  sqlQuery = paste("SELECT MIN(ccFraudLogitScore) AS minVal,",
                  "MAX(ccFraudLogitScore) AS maxVal FROM ccScoreOutput"),
  connectionString = sqlConnString)
```

From this example, you can see how easy it is to use `RxSqlServerData` data source objects to define arbitrary datasets based on SQL queries, functions, or stored

procedures, and then use those in your R code. The variable does not store the actual values, just the data source definition; the query is executed to generate the values only when you use it in a function like **rxImport**.

2. Call the **rxImport** function to put the values in a data frame that can be shared across compute contexts.

```
R

minMaxVals <- rxImport(sqlMinMax)
minMaxVals <- as.vector(unlist(minMaxVals))
```

Results

```
R

> minMaxVals

[1] -23.970256  9.786345
```

3. Now that the maximum and minimum values are available, use the values to create another data source for the generated scores.

```
R

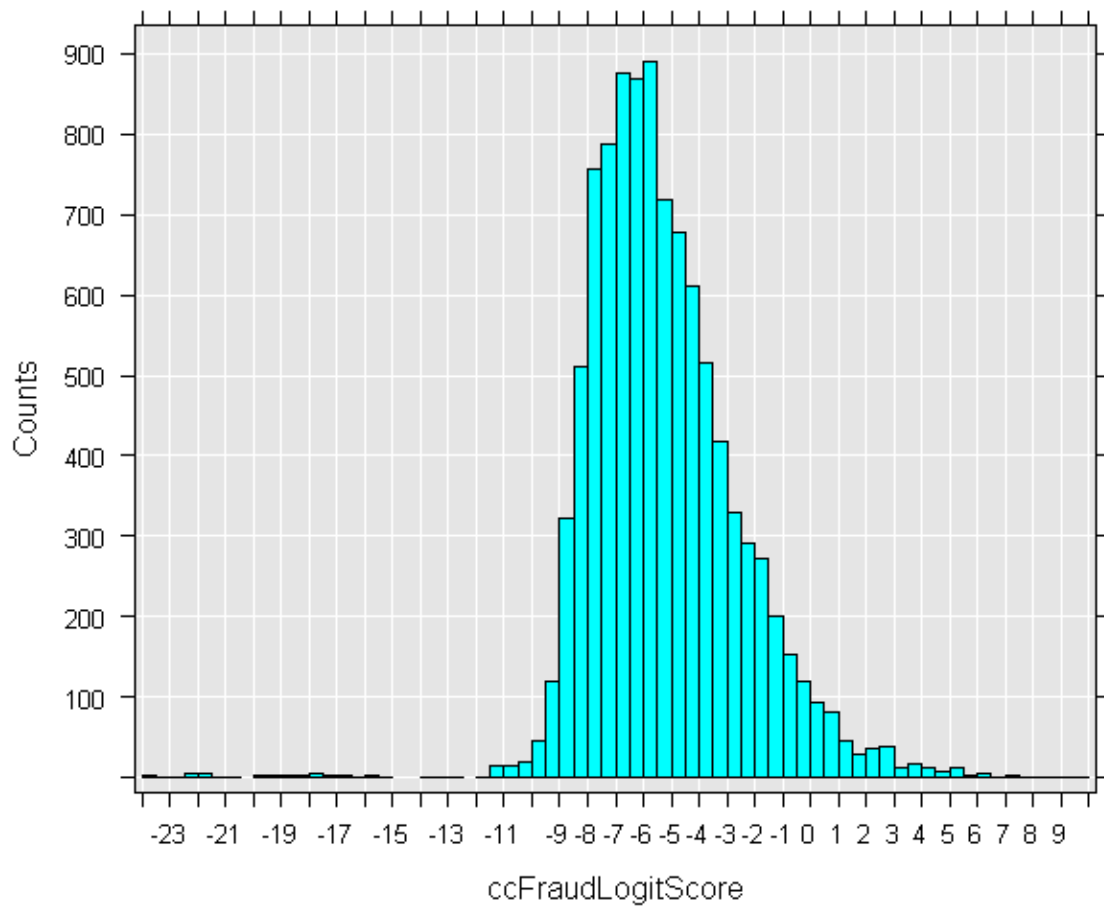
sqlOutScoreDS <- RxSqlServerData(sqlQuery = "SELECT ccFraudLogitScore
FROM ccScoreOutput",
  connectionString = sqlConnString,
  rowsPerRead = sqlRowsPerRead,
  colInfo = list(ccFraudLogitScore = list(
    low = floor(minMaxVals[1]),
    high = ceiling(minMaxVals[2]) ) ) )
```

4. Use the data source object `sqlOutScoreDS` to get the scores, and compute and display a histogram. Add the code to set the compute context if needed.

```
R

# rxSetComputeContext(sqlCompute)
rxHistogram(~ccFraudLogitScore, data = sqlOutScoreDS)
```

Results



Next steps

Transform data using R

Transform data using R (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 9 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

In this tutorial, you'll learn about the **RevoScaleR** functions for transforming data at various stages of your analysis.

- ✓ Use **rxDataStep** to create and transform a data subset
- ✓ Use **rxImport** to transform in-transit data to or from an XDF file or an in-memory data frame during import

Although not specifically for data movement, the functions **rxSummary**, **rxCube**, **rxLinMod**, and **rxLogit** all support data transformations.

Use rxDataStep to transform variables

The **rxDataStep** function processes data one chunk at a time, reading from one data source and writing to another. You can specify the columns to transform, the transformations to load, and so forth.

To make this example interesting, let's use a function from another R package to transform the data. The **boot** package is one of the "recommended" packages, meaning that **boot** is included with every distribution of R, but is not loaded automatically on start-up. Therefore, the package should already be available on the SQL Server instance configured for R integration.

From the **boot** package, use the function **inv.logit**, which computes the inverse of a logit. That is, the **inv.logit** function converts a logit back to a probability on the [0,1] scale.

Tip

Another way to get predictions in this scale would be to set the *type* parameter to **response** in the original call to **rxPredict**.

1. Start by creating a data source to hold the data destined for the table, `ccScoreOutput`.

R

```
sqlOutScoreDS <- RxSqlServerData( table = "ccScoreOutput",  
  connectionString = sqlConnString, rowsPerRead = sqlRowsPerRead )
```

2. Add another data source to hold the data for the table `ccScoreOutput2`.

R

```
sqlOutScoreDS2 <- RxSqlServerData( table = "ccScoreOutput2",  
  connectionString = sqlConnString, rowsPerRead = sqlRowsPerRead )
```

In the new table, store all the variables from the previous `ccScoreOutput` table, plus the newly created variable.

3. Set the compute context to the SQL Server instance.

R

```
rxSetComputeContext(sqlCompute)
```

4. Use the function `rxSqlServerTableExists` to check whether the output table `ccScoreOutput2` already exists; and if so, use the function `rxSqlServerDropTable` to delete the table.

R

```
if (rxSqlServerTableExists("ccScoreOutput2"))  
  rxSqlServerDropTable("ccScoreOutput2")
```

5. Call the `rxDataStep` function, and specify the desired transforms in a list.

R

```
rxDataStep(inData = sqlOutScoreDS,  
  outFile = sqlOutScoreDS2,  
  transforms = list(ccFraudProb = inv.logit(ccFraudLogitScore)),  
  transformPackages = "boot",  
  overwrite = TRUE)
```

When you define the transformations that are applied to each column, you can also specify any additional R packages that are needed to perform the transformations. For more information about the types of transformations that you can perform, see [How to transform and subset data using RevoScaleR](#).

6. Call `rxGetVarInfo` to view a summary of the variables in the new data set.

```
R
```

```
rxGetVarInfo(sqlOutScoreDS2)
```

Results

```
R
```

```
Var 1: ccFraudLogitScore, Type: numeric  
Var 2: state, Type: character  
Var 3: gender, Type: character  
Var 4: cardholder, Type: character  
Var 5: balance, Type: integer  
Var 6: numTrans, Type: integer  
Var 7: numIntlTrans, Type: integer  
Var 8: creditLine, Type: integer  
Var 9: ccFraudProb, Type: numeric
```

The original logit scores are preserved, but a new column, `ccFraudProb`, has been added, in which the logit scores are represented as values between 0 and 1.

Notice that the factor variables have been written to the table `ccScoreOutput2` as character data. To use them as factors in subsequent analyses, use the parameter `collInfo` to specify the levels.

Next steps

[Load data into memory using rxImport](#)

Load data into memory using rxImport (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 10 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

In this tutorial, you'll learn how to get data from SQL Server, and then use the `rxImport` function to put the data of interest into a local file. That way, you can analyze it in the local compute context repeatedly, without having to re-query the database.

The `rxImport` function can be used to move data from a data source into a data frame in session memory, or into an XDF file on disk. If you don't specify a file as destination, data is put into memory as a data frame.

Extract a subset of data from SQL Server to local memory

You've decided that you want to examine only the high risk individuals in more detail. The source table in SQL Server is big, so you want to get the information about just the high-risk customers. You then load that data into a data frame in the memory of the local workstation.

1. Reset the compute context to your local workstation.

```
R  
  
rxSetComputeContext("local")
```

2. Create a new SQL Server data source object, providing a valid SQL statement in the `sqlQuery` parameter. This example gets a subset of the observations with the highest risk scores. That way, only the data you really need is put in local memory.

```
R  
  
sqlServerProbDS \<- RxSqlServerData(  
  sqlQuery = paste("SELECT * FROM ccScoreOutput2",  
    "WHERE (ccFraudProb > .99)"),  
  connectionString = sqlConnString)
```


3. Call the function `rxImport` to read the data into a data frame in the local R session.

```
R  
  
highRisk <- rxImport(sqlServerProbDS)
```

If the operation was successful, you should see a status message like this one:
"Rows Read: 35, Total Rows Processed: 35, Total Chunk Time: 0.036 seconds"

4. Now that the high-risk observations are in an in-memory data frame, you can use various R functions to manipulate the data frame. For example, you can order customers by their risk score, and print a list of the customers who pose the highest risk.

```
R  
  
orderedHighRisk <- highRisk[order(-highRisk$ccFraudProb),]  
row.names(orderedHighRisk) <- NULL  
head(orderedHighRisk)
```

Results

```
R  
  
ccFraudLogitScore  state gender cardholder balance numTrans numIntlTrans  
creditLine ccFraudProb1  
9.786345 SD Male Principal 23456 25 5 75  
0.99994382  
9.433040 FL Female Principal 20629 24 28 75  
0.99992003  
8.556785 NY Female Principal 19064 82 53 43  
0.99980784  
8.188668 AZ Female Principal 19948 29 0 75  
0.99972235  
7.551699 NY Female Principal 11051 95 0 75  
0.99947516  
7.335080 NV Male Principal 21566 4 6 75  
0.9993482
```

More about rxImport

You can use `rxImport` not just to move data, but to transform data in the process of reading it. For example, you can specify the number of characters for fixed-width columns, provide a description of the variables, set levels for factor columns, and even create new levels to use after importing.

The **rxImport** function assigns variable names to the columns during the import process, but you can indicate new variable names by using the *colInfo* parameter, or change data types using the *colClasses* parameter.

By specifying additional operations in the *transforms* parameter, you can do elementary processing on each chunk of data that is read.

Next steps

Create new SQL Server table using rxDataStep

Create new SQL Server table using rxDataStep (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 11 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

In this tutorial, you'll learn how to move data between in-memory data frames, the SQL Server context, and local files.

Note

This tutorial uses a different data set. The Airline Delays dataset is a public dataset that is widely used for machine learning experiments. The data files used in this example are available in the same directory as other product samples.

Load data from a local XDF file

In the first half of this tutorial series, you used the `RxTextData` function to import data into R from a text file, and then used the `RxDataStep` function to move the data into SQL Server.

This tutorial takes a different approach, and uses data from a file saved in the [XDF format](#). After doing some lightweight transformations on the data using the XDF file, you save the transformed data into a new SQL Server table.

What is XDF?

The XDF format is an XML standard developed for high-dimensional data. It is a binary file format with an R interface that optimizes row and column processing and analysis. You can use it for moving data and to store subsets of data that are useful for analysis.

1. Set the compute context to the local workstation. **DDL permissions are needed for this step.**

```
rxSetComputeContext("local")
```

2. Define a new data source object using the **RxXdfData** function. To define an XDF data source, specify the path to the data file.

You could specify the path to the file using a text variable. However, in this case, there's a handy shortcut, which is to use the **rxGetOption** function and get the file (**AirlineDemoSmall.xdf**) from the sample data directory.

```
R
```

```
xdfAirDemo <- RxXdfData(file.path(rxGetOption("sampleDataDir"),  
"AirlineDemoSmall.xdf"))
```

3. Call **rxGetVarInfo** on the in-memory data to view a summary of the dataset.

```
R
```

```
rxGetVarInfo(xdfAirDemo)
```

Results

```
R
```

```
Var 1: ArrDelay, Type: integer, Low/High: (-86, 1490)  
Var 2: CRSDepTime, Type: numeric, Storage: float32, Low/High: (0.0167,  
23.9833)  
Var 3: DayOfWeek 7 factor levels: Monday Tuesday Wednesday Thursday Friday  
Saturday Sunday
```

ⓘ Note

Did you notice that you did not need to call any other functions to load the data into the XDF file, and could call **rxGetVarInfo** on the data immediately? That's because XDF is the default interim storage method for **RevoScaleR**. In addition to XDF files, the **rxGetVarInfo** function now supports multiple source types.

Move contents to SQL Server

With the XDF data source created in the local R session, you can now move this data into a database table, storing *DayOfWeek* as an integer with values from 1 to 7.

1. Define a SQL Server data source object, specifying a table to contain the data, and connection to the remote server.

R

```
sqlServerAirDemo <- RxSqlServerData(table = "AirDemoSmallTest",  
connectionString = sqlConnString)
```

2. As a precaution, include a step that checks whether a table with the same name already exists, and delete the table if it exists. An existing table of the same names prevents a new one from being created.

R

```
if (rxSqlServerTableExists("AirDemoSmallTest", connectionString =  
sqlConnString)) rxSqlServerDropTable("AirDemoSmallTest",  
connectionString = sqlConnString)
```

3. Load the data into the table using `rxDataStep`. This function moves data between two already defined data sources and can optionally transform the data en route.

R

```
rxDataStep(inData = xdfAirDemo, outFile = sqlServerAirDemo,  
transforms = list( DayOfWeek = as.integer(DayOfWeek),  
rowNum = .rxStartRow : (.rxStartRow + .rxNumRows - 1) ),  
overwrite = TRUE )
```

This is a fairly large table, so wait until you see a final status message like this one:
Rows Read: 200000, Total Rows Processed: 600000.

Load data from a SQL table

Once data exists in the table, you can load it by using a simple SQL query.

1. Create a new SQL Server data source. The input is a query on the new table you just created and loaded with data. This definition adds factor levels for the *DayOfWeek* column, using the *collInfo* argument to `RxSqlServerData`.

R

```
sqlServerAirDemo2 <- RxSqlServerData(  
sqlQuery = "SELECT * FROM AirDemoSmallTest",  
connectionString = sqlConnString,  
rowsPerRead = 50000,
```

```
colInfo = list(DayOfWeek = list(type = "factor", levels =  
as.character(1:7))))
```

2. Call `rxSummary` once more to review a summary of the data in your query.

```
R
```

```
rxSummary(~., data = sqlServerAirDemo2)
```

Next steps

[Perform chunking analysis using `rxDataStep`](#)

Perform chunking analysis using rxDataStep (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 12 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

In this tutorial, you'll use the **rxDataStep** function to process data in chunks, rather than requiring that the entire dataset be loaded into memory and processed at one time, as in traditional R. The **rxDataStep** functions reads the data in chunk, applies R functions to each chunk of data in turn, and then saves the summary results for each chunk to a common SQL Server data source. When all data has been read, the results are combined.

Tip

For this tutorial, you compute a contingency table by using the **table** function in R. This example is meant for instructional purposes only.

If you need to tabulate real-world data sets, we recommend that you use the **rxCrossTabs** or **rxCube** functions in **RevoScaleR**, which are optimized for this sort of operation.

Partition data by values

1. Create a custom R function that calls the R **table** function on each chunk of data, and name the new function **ProcessChunk**.

```
R

ProcessChunk <- function( dataList ) {
  # Convert the input list to a data frame and compute contingency table
  chunkTable <- table(as.data.frame(dataList))

  # Convert table output to a data frame with a single row
  varNames <- names(chunkTable)
  varValues <- as.vector(chunkTable)
  dim(varValues) <- c(1, length(varNames))
}
```

```

chunkDF <- as.data.frame(varValues)
names(chunkDF) <- varNames

# Return the data frame, which has a single row
return( chunkDF )
}

```

2. Set the compute context to the server.

```

R

rxSetComputeContext(sqlCompute)

```

3. Define a SQL Server data source to hold the data you're processing. Start by assigning a SQL query to a variable. Then, use that variable in the *sqlQuery* argument of a new SQL Server data source.

```

R

dayQuery <- "SELECT DayOfWeek FROM AirDemoSmallTest"
inDataSource <- RxSqlServerData(sqlQuery = dayQuery,
  connectionString = sqlConnString,
  rowsPerRead = 50000,
  colInfo = list(DayOfWeek = list(type = "factor",
    levels = as.character(1:7))))

```

4. Optionally, you can run **rxGetVarInfo** on this data source. At this point, it contains a single column: *Var 1: DayOfWeek, Type: factor, no factor levels available*

5. Before applying this factor variable to the source data, create a separate table to hold the intermediate results. Again, you just use the **RxSqlServerData** function to define the data, making sure to delete any existing tables of the same name.

```

R

iroDataSource = RxSqlServerData(table = "iroResults",
  connectionString = sqlConnString)
# Check whether the table already exists.
if (rxSqlServerTableExists(table = "iroResults", connectionString =
  sqlConnString)) { rxSqlServerDropTable( table = "iroResults",
  connectionString = sqlConnString) }

```

6. Call the custom function **ProcessChunk** to transform the data as it is read, by using it as the *transformFunc* argument to the **rxDataStep** function.

```

R

```



```
rxDataStep( inData = inDataSource, outFile = iroDataSource,  
transformFunc = ProcessChunk, overwrite = TRUE)
```

- To view the intermediate results of **ProcessChunk**, assign the results of **rxImport** to a variable, and then output the results to the console.

R

```
iroResults <- rxImport(iroDataSource)  
iroResults
```

Partial results

Row #	1	2	3	4	5	6	7
1	8228	8924	6916	6932	6944	5602	6454
2	8321	5351	7329	7411	7409	6487	7692

- To compute the final results across all chunks, sum the columns, and display the results in the console.

R

```
finalResults <- colSums(iroResults)  
finalResults
```

Results

1	2	3	4	5	6	7
97975	77725	78875	81304	82987	86159	94975

- To remove the intermediate results table, make a call to **rxSqlServerDropTable**.

R

```
rxSqlServerDropTable( table = "iroResults", connectionString =  
sqlConnString)
```

Next steps

Move data between SQL Server and XDF file (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 13 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

In this tutorial, you'll learn how to use an XDF file to transfer data between remote and local compute contexts. Storing the data in an XDF file allows you to perform transformations on the data.

When you're done, you use the data in the file to create a new SQL Server table. The function [rxDataStep](#) can apply transformations to the data and performs the conversion between data frames and .xdf files.

Create a SQL Server table from an XDF file

For this exercise, you use the credit card fraud data again. In this scenario, you've been asked to do some extra analysis on users in the states of California, Oregon, and Washington. To be more efficient, you've decided to store data for only these states on your local computer, and work with only the variables gender, cardholder, state, and balance.

1. Re-use the `stateAbb` variable you created earlier to identify the levels to include, and write them to a new variable, `statesToKeep`.

R

```
statesToKeep <- sapply(c("CA", "OR", "WA"), grep, stateAbb)
statesToKeep
```

Results

CA	OR	WA
5	38	48

2. Define the data you want to bring over from SQL Server, using a Transact-SQL query. Later you use this variable as the *inData* argument for `rxImport`.
-

R

```
importQuery <- paste("SELECT gender,cardholder,balance,state FROM",  
sqlFraudTable, "WHERE (state = 5 OR state = 38 OR state = 48)")
```

Make sure there are no hidden characters such as line feeds or tabs in the query.

3. Next, define the columns to use when working with the data in R. For example, in the smaller data set, you need only three factor levels, because the query returns data for only three states. Apply the `statesToKeep` variable to identify the correct levels to include.

R

```
importColumnInfo <- list(  
  gender = list( type = "factor", levels = c("1", "2"), newLevels =  
    c("Male", "Female")),  
  cardholder = list( type = "factor", levels = c("1", "2"),  
    newLevels = c("Principal", "Secondary")),  
  state = list( type = "factor", levels =  
    as.character(statesToKeep), newLevels = names(statesToKeep))  
)
```

4. Set the compute context to **local**, because you want all the data available on your local computer.

R

```
rxSetComputeContext("local")
```

The `rxImport` function can import data from any supported data source to a local XDF file. Using a local copy of the data is convenient when you want to do many different analyses on the data, but want to avoid running the same query over and over.

5. Create the data source object by passing the variables previously defined as arguments to **RxSqlServerData**.

R

```
sqlServerImportDS <- RxSqlServerData(  
  connectionString = sqlConnString,  
  sqlQuery = importQuery,  
  colInfo = importColumnInfo)
```

6. Call `rxImport` to write the data to a file named `ccFraudSub.xdf`, in the current working directory.

R

```
localDS <- rxImport(inData = sqlServerImportDS,  
  outFile = "ccFraudSub.xdf",  
  overwrite = TRUE)
```

The `localDS` object returned by the `rxImport` function is a light-weight `RxXdfData` data source object that represents the `ccFraud.xdf` data file stored locally on disk.

7. Call `rxGetVarInfo` on the XDF file to verify that the data schema is the same.

R

```
rxGetVarInfo(data = localDS)
```

Results

R

```
rxGetVarInfo(data = localDS)  
Var 1: gender, Type: factor, no factor levels available  
Var 2: cardholder, Type: factor, no factor levels available  
Var 3: balance, Type: integer, Low/High: (0, 22463)  
Var 4: state, Type: factor, no factor levels available
```

8. You can now call various R functions to analyze the `localDS` object, just as you would with the source data on SQL Server. For example, you might summarize by gender:

R

```
rxSummary(~gender + cardholder + balance + state, data = localDS)
```

Next steps

This tutorial concludes the multi-part tutorial series on **RevoScaleR** and SQL Server. It introduced you to numerous data-related and computational concepts, giving you a foundation for moving forward with your own data and project requirements.

To deepen your knowledge of **RevoScaleR**, you can return to the R tutorials list to step through any exercises you might have missed. Alternatively, review the How-to articles in the table of contents for information about general tasks.

[R Tutorials for SQL Server](#)

Run custom R functions on SQL Server using rxExec (SQL Server and RevoScaleR tutorial)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This is tutorial 14 of the [RevoScaleR tutorial series](#) on how to use [RevoScaleR functions](#) with SQL Server.

In this tutorial, you'll use simulated data to demonstrate execution of a custom R function that runs on a remote server.

You can run custom R functions in the context of SQL Server by passing your function via [rxExec](#), assuming that any libraries your script requires are also installed on the server and those libraries are compatible with the base distribution of R.

The [rxExec](#) function in [RevoScaleR](#) provides a mechanism for running any R script you require. Additionally, [rxExec](#) is able to explicitly distribute work across multiple cores in a single server, adding scale to scripts that are otherwise limited to the resource constraints of the native R engine.

Prerequisites

- [SQL Server Machine Learning Services \(with R\)](#) or [SQL Server 2016 R Services \(in-Database\)](#)
- [Database permissions](#) and a SQL Server database user login
- [A development workstation with the RevoScaleR libraries](#)

The R distribution on the client workstation provides a built-in **Rgui** tool that you can use to run the R script in this tutorial. You can also use an IDE such as RStudio or R Tools for Visual Studio.

Create the remote compute context

Run the following R commands on a client workstation. For example, you are using **Rgui**, start it from this location: C:\Program Files\Microsoft\R Client\R_SERVER\bin\x64.

1. Specify the connection string for the SQL Server instance where computations are performed. The server must be configured for R integration. The database name is not used in this exercise, but the connection string requires one. If you have a test or sample database, you can use that.

Using a SQL login

R

```
sqlConnString <- "Driver=SQL Server;Server=<SQL-Server-instance-name>;  
Database=<database-name>;Uid=<SQL-user-name>;Pwd=<password>"
```

Using Windows authentication

R

```
sqlConnString <- "Driver=SQL Server;Server=<SQL-Server-instance-  
name>;Database=<database-name>;Trusted_Connection=True"
```

2. Create a remote compute context to the SQL Server instance referenced in the connection string.

R

```
sqlCompute <- RxInSqlServer(connectionString = sqlConnString)
```

3. Activate the compute context and then return the object definition as a confirmation step. You should see the properties of the compute context object.

R

```
rxSetComputeContext(sqlCompute)  
rxGetComputeContext()
```

Create the custom function

In this exercise, you will create a custom R function that simulates a common casino consisting of rolling a pair of dice. Rules of the game determine a win or loss outcome:

- Roll a 7 or 11 on your initial roll, you win.
- Roll 2, 3, or 12, you lose.
- Roll a 4, 5, 6, 8, 9, or 10, that number becomes your point, and you continue rolling until you either roll your point again (in which case you win) or roll a 7, in which

case you lose.

The game is easily simulated in R, by creating a custom function, and then running it many times.

1. Create the custom function using the following R code:

```
R

rollDice <- function()
{
  result <- NULL
  point <- NULL
  count <- 1
  while (is.null(result))
  {
    roll <- sum(sample(6, 2, replace=TRUE))

    if (is.null(point))
    { point <- roll }
    if (count == 1 && (roll == 7 || roll == 11))
    { result <- "Win" }
    else if (count == 1 && (roll == 2 || roll == 3 || roll ==
12))
    { result <- "Loss" }
    else if (count > 1 && roll == 7 )
    { result <- "Loss" }
    else if (count > 1 && point == roll)
    { result <- "Win" }
    else { count <- count + 1 }
  }
  result
}
```

2. Simulate a single game of dice by running the function.

```
R

rollDice()
```

Did you win or lose?

Now that you have an operational script, let's see how you can use `rxExec` to run the function multiple times to create a simulation that helps determine the probability of a win.

Pass `rollDice()` in `rxExec`

To run an arbitrary function in the context of a remote SQL Server, call the `rxExec` function.

1. Call the custom function as an argument to `rxExec`, together with other parameters that modify the simulation.

R

```
sqlServerExec <- rxExec(rollDice, timesToRun=20, RNGseed="auto")
length(sqlServerExec)
```

- Use the *timesToRun* argument to indicate how many times the function should be executed. In this case, you roll the dice 20 times.
- The arguments *RNGseed* and *RNGkind* can be used to control random number generation. When *RNGseed* is set to **auto**, a parallel random number stream is initialized on each worker.

2. The `rxExec` function creates a list with one element for each run; however, you won't see much happening until the list is complete. When all the iterations are complete, the line starting with **length** will return a value.

You can then go to the next step to get a summary of your win-loss record.

3. Convert the returned list to a vector using R's **unlist** function, and summarize the results using the **table** function.

R

```
table(unlist(sqlServerExec))
```

Your results should look something like this:

Loss Win 12 8

Conclusion

Although this exercise is simplistic, it demonstrates an important mechanism for integrating arbitrary R functions in R script running on SQL Server. To summarize the key points that make this technique possible:

- SQL Server must be configured for machine learning and R integration: [SQL Server Machine Learning Services](#) with the R feature, or [SQL Server 2016 R Services \(in-Database\)](#).

- Open-source or third-party libraries used in your function, including any dependencies, must be installed on SQL Server. For more information, see [Install new R packages](#).
- Moving script from a development environment to a hardened production environment can introduce firewall and network restrictions. Test carefully to make sure your script is able to perform as expected.

Next steps

For a more complex example of using `rxExec`, see this article: [Coarse grain parallelism with foreach and rxExec](#) [↗](#)

Airline flight arrival demo data for SQL Server Python and R tutorials

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

In this exercise, create a SQL Server database to store imported data from R or Python built-in Airline demo data sets. R and Python distributions provide equivalent data, which you can import to a SQL Server database using Management Studio.

To complete this exercise, you should have [SQL Server Management Studio](#) or another tool that can run T-SQL queries.

Tutorials and quickstarts using this data set include the following:

- [Create a Python model using revoscalepy](#)

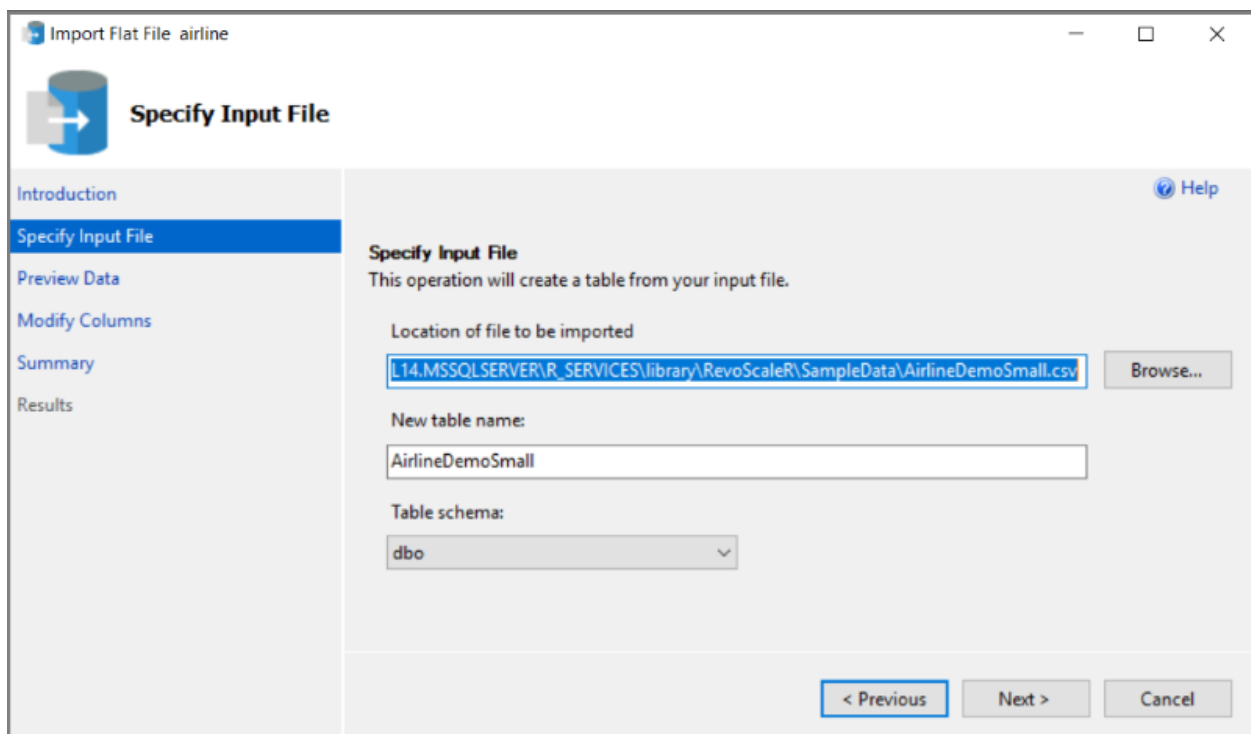
Create the database

1. Start SQL Server Management Studio, connect to a database engine instance that has R or Python integration.
2. In Object Explorer, right-click **Databases** and create a new database called **flightdata**.
3. Right-click **flightdata**, click **Tasks**, click **Import Flat File**.
4. Open the AirlineDemoData.csv file provided in the R or Python distribution, depending on which language you installed.

For R, look for **AirlineDemoSmall.csv** at C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library\RevoScaleR\SampleData

For Python, look for **AirlineDemoSmall.csv** at C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES\Lib\site-packages\revoscalepy\data\sample_data

When you select the file, default values are filled in for table name and schema.



Click through the remaining pages, accepting the defaults, to import the data.

Query the data

As a validation step, run a query to confirm the data was uploaded.

1. In Object Explorer, under Databases, right-click the **flightdata** database, and start a new query.
2. Run some simple queries:

SQL

```
SELECT TOP(10) * FROM AirlineDemoSmall;  
SELECT COUNT(*) FROM AirlineDemoSmall;
```


Next steps

In the following lesson, you will create a linear regression model based on this data.

- [Create a Python model using revoscalepy](#)

Iris demo data for Python and R tutorials with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

In this exercise, create a database to store data from the [Iris flower data set](#) and models based on the same data. Iris data is included in both the R and Python distributions, and is used in machine learning tutorials for SQL machine learning.

To complete this exercise, you should have [SQL Server Management Studio](#) or another tool that can run T-SQL queries.

Tutorials and quickstarts using this data set include the following:

- [Quickstart: Create and score a predictive model in Python](#)

Create the database

1. Start SQL Server Management Studio, and open a new **Query** window.
2. Create a new database for this project, and change the context of your **Query** window to use the new database.

SQL

```
CREATE DATABASE irissql
GO
USE irissql
GO
```

3. Add some empty tables: one to store the data, and one to store the trained models. The `iris_models` table is used for storing serialized models generated in other exercises.

The following code creates the table for the training data.

SQL

```
DROP TABLE IF EXISTS iris_data;
GO
CREATE TABLE iris_data (
    id INT NOT NULL IDENTITY PRIMARY KEY
    , "Sepal.Length" FLOAT NOT NULL, "Sepal.Width" FLOAT NOT NULL
```

```
, "Petal.Length" FLOAT NOT NULL, "Petal.Width" FLOAT NOT NULL
, "Species" VARCHAR(100) NOT NULL, "SpeciesId" INT NOT NULL
);
```

4. Run the following code to create the table used for storing the trained model. To save Python (or R) models in SQL Server, they must be serialized and stored in a column of type `varbinary(max)`.

SQL

```
DROP TABLE IF EXISTS iris_models;
GO

CREATE TABLE iris_models (
    model_name VARCHAR(50) NOT NULL DEFAULT('default model') PRIMARY KEY,
    model VARBINARY(MAX) NOT NULL
);
GO
```

In addition to the model contents, typically, you would also add columns for other useful metadata, such as the model's name, the date it was trained, the source algorithm and parameters, source data, and so forth. For now we'll keep it simple and use just the model name.

Populate the table

You can obtain built-in Iris data from either R or Python. You can use Python or R to load the data into a data frame, and then insert it into a table in the database. Moving training data from an external session into a table is a multistep process:

- Design a stored procedure that gets the data you want.
- Execute the stored procedure to actually get the data.
- Construct an INSERT statement to specify where the retrieved data should be saved.

1. On systems with Python integration, create the following stored procedure that uses Python code to load the data.

SQL

```
CREATE PROCEDURE get_iris_dataset
AS
BEGIN
EXEC sp_execute_external_script @language = N'Python',
@script = N'
from sklearn import datasets
```

```

iris = datasets.load_iris()
iris_data = pandas.DataFrame(iris.data)
iris_data["Species"] = pandas.Categorical.from_codes(iris.target,
iris.target_names)
iris_data["SpeciesId"] = iris.target
',
@input_data_1 = N'',
@output_data_1_name = N'iris_data'
WITH RESULT SETS (("Sepal.Length" float not null, "Sepal.Width" float
not null, "Petal.Length" float not null, "Petal.Width" float not null,
"Species" varchar(100) not null, "SpeciesId" int not null));
END;
GO

```

When you run this code, you should get the message "Commands completed successfully." All this means is that the stored procedure has been created according to your specifications.

2. Alternatively, on systems having R integration, create a procedure that uses R instead.

SQL

```

CREATE PROCEDURE get_iris_dataset
AS
BEGIN
EXEC sp_execute_external_script @language = N'R',
@script = N'
library(RevoScaleR)
data(iris)
iris$SpeciesID <- c(unclass(iris$Species))
iris_data <- iris
',
@input_data_1 = N'',
@output_data_1_name = N'iris_data'
WITH RESULT SETS (("Sepal.Length" float not null, "Sepal.Width" float
not null, "Petal.Length" float not null, "Petal.Width" float not null,
"Species" varchar(100) not null, "SpeciesId" int not null));
END;
GO

```

3. To actually populate the table, run the stored procedure and specify the table where the data should be written. When run, the stored procedure executes the Python or R code, which loads the built-in Iris data set, and then inserts the data into the `iris_data` table.

SQL

```
INSERT INTO iris_data ("Sepal.Length", "Sepal.Width", "Petal.Length",  
"Petal.Width", "Species", "SpeciesId")  
EXEC dbo.get_iris_dataset;
```

If you're new to T-SQL, be aware that the INSERT statement only adds new data; it won't check for existing data, or delete and rebuild the table. To avoid getting multiple copies of the same data in a table, you can run this statement first:

`TRUNCATE TABLE iris_data`. The T-SQL `TRUNCATE TABLE` statement deletes existing data but keeps the structure of the table intact.

Query the data

As a validation step, run a query to confirm the data was uploaded.

1. In Object Explorer, under Databases, right-click the **irissql** database, and start a new query.
2. Run some simple queries:

SQL

```
SELECT TOP(10) * FROM iris_data;  
SELECT COUNT(*) FROM iris_data;
```

Next steps

In the following quickstart, you will create a machine learning model and save it to a table, and then use the model to generate predicted outcomes.

- [Quickstart:Create and score a predictive model in Python](#)

NYC Taxi demo data for SQL Server Python and R tutorials

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

This article explains how to set up a sample database consisting of public data from the [New York City Taxi and Limousine Commission](#). This data is used in several R and Python tutorials for in-database analytics on SQL Server. To make the sample code run quicker, we created a representative 1% sampling of the data. On your system, the database backup file is slightly over 90 MB, providing 1.7 million rows in the primary data table.

To complete this exercise, you should have [SQL Server Management Studio \(SSMS\)](#) or another tool that can restore a database backup file and run T-SQL queries.

Tutorials and quickstarts using this data set include the following:

- [Learn in-database analytics using R in SQL Server](#)
- [Learn in-database analytics using Python in SQL Server](#)

Download files

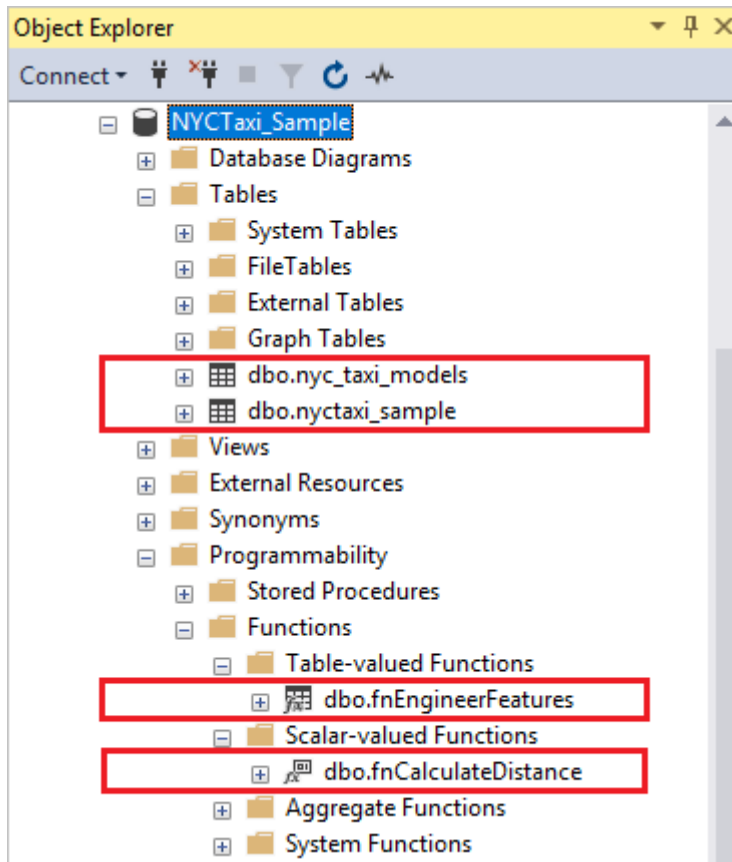
The sample database is a SQL Server 2016 BAK file hosted by Microsoft. You can restore it on SQL Server 2016 and later. File download begins immediately when you open the link.

File size is approximately 90 MB.

1. Download the [NYCTaxi_Sample.bak](#) database backup file.
2. Copy the file to `C:\Program files\Microsoft SQL Server\MSSQL-instance-name\MSSQL\Backup` or similar path, for your instance's default `Backup` folder.
3. In SSMS, right-click **Databases** and select **Restore Files and File Groups**.
4. Enter `NYCTaxi_Sample` as the database name.
5. Select **From device** and then open the file selection page to select the `NYCTaxi_Sample.bak` backup file. Select **Add** to select `NYCTaxi_Sample.bak`.
6. Select the **Restore** checkbox and select **OK** to restore the database.

Review database objects

Confirm the database objects exist on the SQL Server instance using SQL Server Management Studio. You should see the database, tables, functions, and stored procedures.



Objects in NYCTaxi_Sample database

The following table summarizes the objects created in the NYC Taxi demo database.

Object name	Object type	Description
NYCTaxi_Sample	database	Creates a database and two tables: <code>dbo.nyctaxi_sample</code> table: Contains the main NYC Taxi dataset. A clustered columnstore index is added to the table to improve storage and query performance. The 1% sample of the NYC Taxi dataset is inserted into this table. <code>dbo.nyc_taxi_models</code> table: Used to persist the trained advanced analytics model.

Object name	Object type	Description
fnCalculateDistance	scalar-valued function	Calculates the direct distance between pickup and dropoff locations. This function is used in Create data features , Train and save a model and Operationalize the R model .
fnEngineerFeatures	table-valued function	Creates new data features for model training. This function is used in Create data features and Operationalize the R model .

Stored procedures are created using R and Python script found in various tutorials. The following table summarizes the stored procedures that you can optionally add to the NYC Taxi demo database when you run script from various lessons.

Stored procedure	Language	Description
RxPlotHistogram	R	Calls the RevoScaleR <code>rxHistogram</code> function to plot the histogram of a variable and then returns the plot as a binary object. This stored procedure is used in Explore and visualize data .
RPlotRHist	R	Creates a graphic using the <code>Hist</code> function and saves the output as a local PDF file. This stored procedure is used in Explore and visualize data .
RxTrainLogitModel	R	Trains a logistic regression model by calling an R package. The model predicts the value of the <code>tipped</code> column, and is trained using a randomly selected 70% of the data. The output of the stored procedure is the trained model, which is saved in the table <code>dbo.nyc_taxi_models</code> . This stored procedure is used in Train and save a model .
RxPredictBatchOutput	R	Calls the trained model to create predictions using the model. The stored procedure accepts a query as its input parameter and returns a column of numeric values containing the scores for the input rows. This stored procedure is used in Predict potential outcomes .
RxPredictSingleRow	R	Calls the trained model to create predictions using the model. This stored procedure accepts a new observation as input, with individual feature values passed as in-line parameters, and returns a value that predicts the outcome for the new observation. This stored procedure is used in Predict potential outcomes .

Query the data

As a validation step, run a query to confirm the data was uploaded.

1. In Object Explorer, under **Databases**, right-click the **NYCTaxi_Sample** database, and start a new query.
2. Run some simple queries:

```
SQL

SELECT TOP(10) * FROM dbo.nyctaxi_sample;
SELECT COUNT(*) FROM dbo.nyctaxi_sample;
```

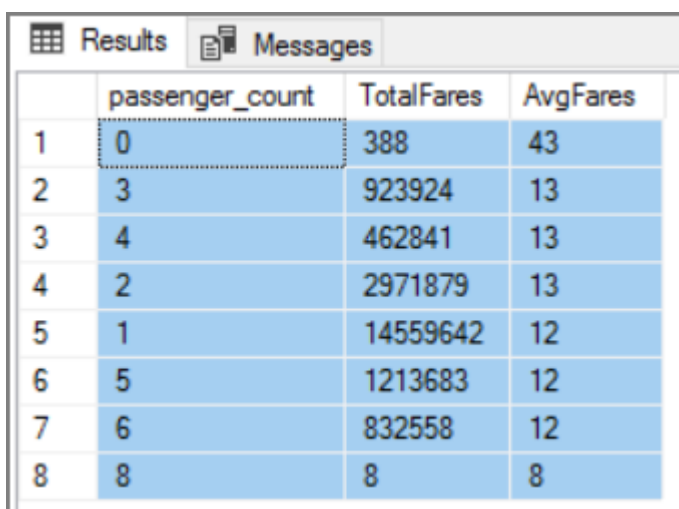
The database contains 1.7 million rows.

3. Within the database is a `dbo.nyctaxi_sample` table that contains the data set. The table has been optimized for set-based calculations with the addition of a [columnstore index](#). Run this statement to generate a quick summary on the table.

```
SQL

SELECT DISTINCT [passenger_count]
    , ROUND (SUM ([fare_amount]),0) as TotalFares
    , ROUND (AVG ([fare_amount]),0) as AvgFares
FROM [dbo].[nyctaxi_sample]
GROUP BY [passenger_count]
ORDER BY AvgFares DESC
```

Results should be similar to those showing in the following screenshot.



	passenger_count	TotalFares	AvgFares
1	0	388	43
2	3	923924	13
3	4	462841	13
4	2	2971879	13
5	1	14559642	12
6	5	1213683	12
7	6	832558	12
8	8	8	8

Next steps

NYC Taxi sample data is now available for hands-on learning.

- [Learn in-database analytics using R in SQL Server](#)
- [Learn in-database analytics using Python in SQL Server](#)

Extensibility architecture in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

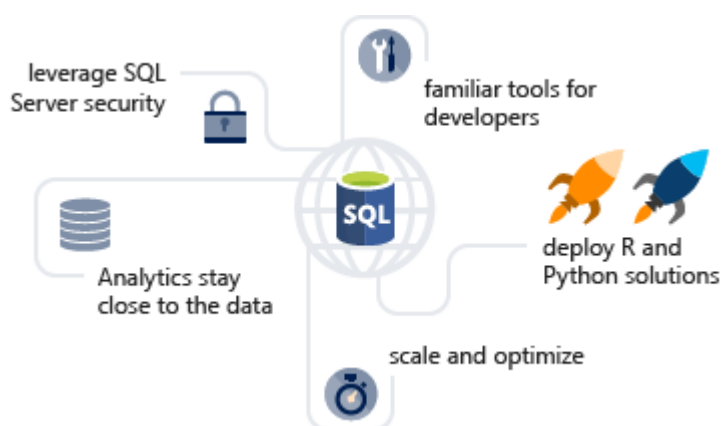
This article describes the architecture of the extensibility framework for running an external Python or R script on SQL server Machine Learning Services. The script executes in a language runtime environment as an extension to the core database engine.

Background

The extensibility framework was introduced in SQL Server 2016 to support the R runtime with [R Services](#). SQL Server 2017 and later has support for Python with [Machine Learning Services](#).

The purpose of the extensibility framework is to provide an interface between SQL Server and data science languages such as R and Python. The goal is to reduce friction when moving data science solutions into production, and protecting data exposed during the development process. By executing a trusted scripting language within a secure framework managed by SQL Server, database administrators can maintain security while allowing data scientists access to enterprise data.

The following diagram visually describes opportunities and benefits of the extensible architecture.



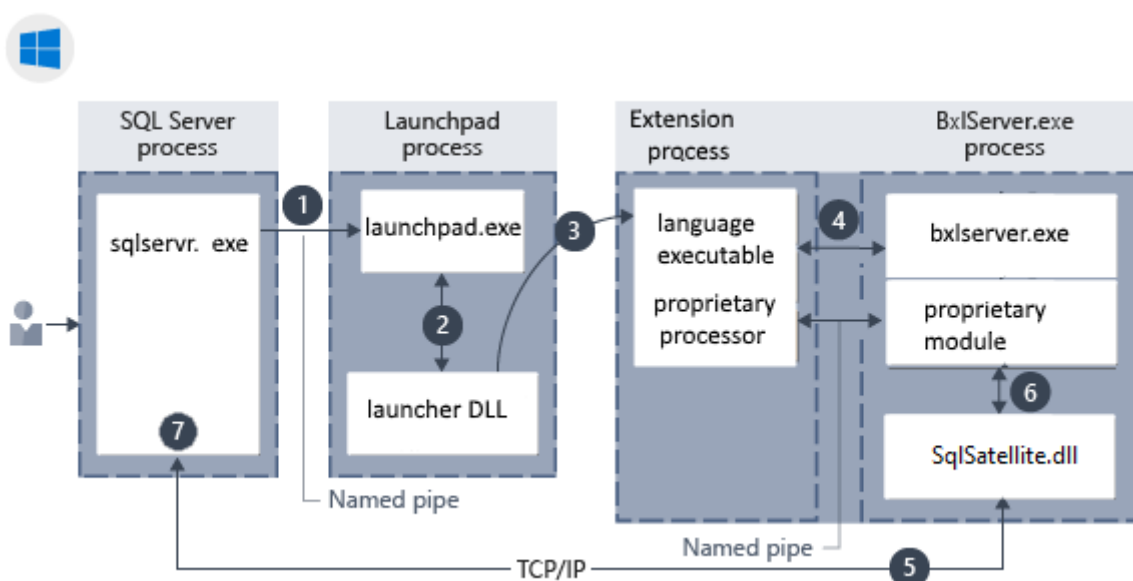
An external script can be run by calling a stored procedure, and the results are returned as tabular results directly to SQL Server. This makes it easy to generate or consume machine learning from any application that can send a SQL query and handle the results.

- External script execution is subject to SQL Server data security. A user running an external script can only access data that is equally available in a SQL query. If a query fails due to insufficient permission, a script run by the same user would also fail for the same reason. SQL Server security is enforced at the table, database, and instance level. Database administrators can manage user access, resources used by external scripts, and external code libraries added to the server.
- Scale and optimization opportunities have a dual basis: gains through the database platform (ColumnStore indexes, [resource governance](#)); and extension-specific gains, for example when Microsoft libraries for R and Python are used for data science models. Whereas R is single-threaded, RevoScaleR functions are multi-threaded, capable of distributing a workload over multiple cores.
- Deployment uses SQL Server methodologies. These can be stored procedures wrapping an external script, embedded SQL, or T-SQL queries calling functions like PREDICT to return results from forecasting models persisted on the server.
- Developers with established skills in specific tools and IDEs can write code in those tools and then port the code to SQL Server.

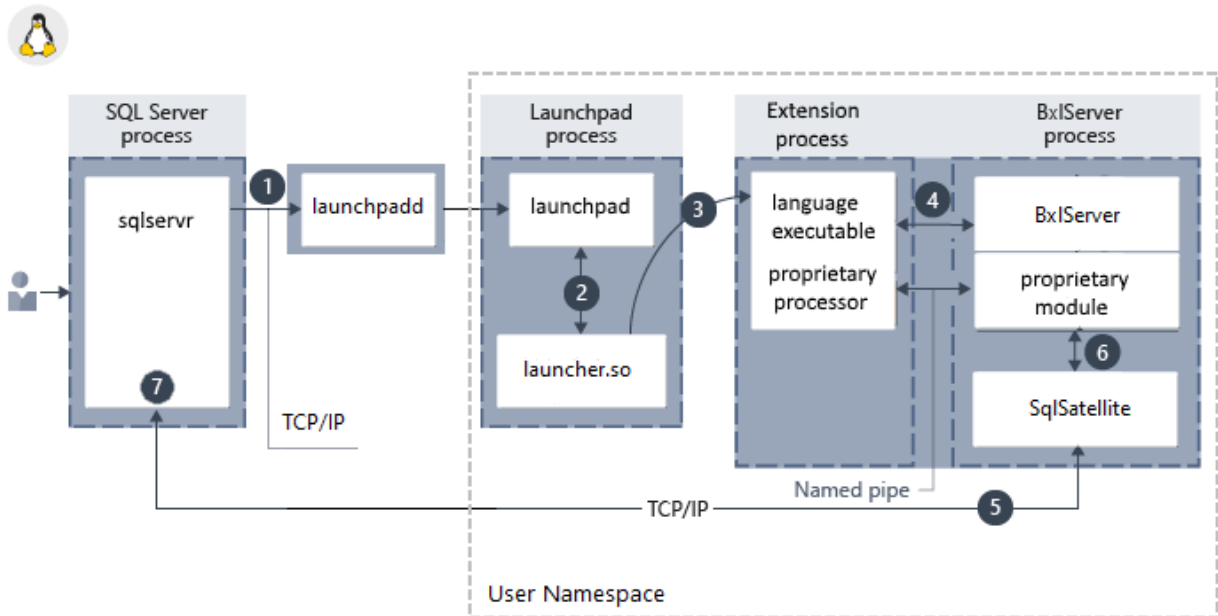
Architecture diagram

The architecture is designed such that external scripts run in a separate process from SQL Server, but with components that internally manage the chain of requests for data and operations on SQL Server. Depending on the version of SQL Server, supported language extensions include [R](#), [Python](#), and third-party languages such as Java and .NET.

Component architecture in Windows:



Component architecture in Linux:



Components include a **launchpad** service used to invoke external runtimes and library-specific logic for loading interpreters and libraries. The launcher loads a language runtime, plus any proprietary modules. For example, if your code includes RevoScaleR functions, a RevoScaleR interpreter is loaded. **BxlServer** and **SQL Satellite** manage communication and data transfer with SQL Server.

In Linux, SQL uses a **launchpadd** service to communicate with a separate launchpad process for each user.

Launchpad

The SQL Server Launchpad is a service that manages and executes external scripts, similar to the way that the full-text indexing and query service launches a separate host for processing full-text queries. The launchpad service can start only trusted launchers that are published by Microsoft, or that have been certified by Microsoft as meeting requirements for performance and resource management.

Trusted launchers	Extension	SQL Server versions
Rlauncher.dll for the R language for Windows	R extension	SQL Server 2016 and later
Pythonlauncher.dll for Python language for Windows	Python extension	SQL Server 2017 and later
Rlauncher.so for the R language for Linux	R extension	SQL Server 2019 and later

Trusted launchers	Extension	SQL Server versions
Pythonlauncher.so for Python language for Linux	Python extension	SQL Server 2019 and later

The SQL Server Launchpad service runs under its own user account. If you change the account that runs launchpad, be sure to do so using SQL Server Configuration Manager, to ensure that changes are written to related files.

In Windows, a separate SQL Server Launchpad service is created for each database engine instance to which you have added SQL Server Machine Learning Services. There is one launchpad service for each database engine instance, so if you have multiple instances with external script support, you will have a launchpad service for each one. A database engine instance is bound to the launchpad service created for it. All invocations of external script in a stored procedure or T-SQL result in the SQL Server service calling the launchpad service created for the same instance.

To execute tasks in a specific supported language, the launchpad gets a secured worker account from the pool, and starts a satellite process to manage the external runtime. Each satellite process inherits the user account of the launchpad and uses that worker account for the duration of script execution. If script uses parallel processes, they are created under the same, single worker account.

In Linux, only one database engine instance is supported and there is one launchpadd service bound to the instance. When a script is executed, the launchpadd service starts a separate launchpad process with the low-privileged user account `mssql_satellite`. Each satellite process inherits the `mssql_satellite` user account of launchpad and uses that for the duration of script execution.

BxlServer and SQL Satellite

BxlServer is an executable provided by Microsoft that manages communication between SQL Server and the language runtime. It creates the Windows job objects for Windows, or the namespaces for Linux, that are used to contain external script sessions. It also provisions secure working folders for each external script job and uses SQL Satellite to manage data transfer between the external runtime and SQL Server. If you run [Process Explorer](#) while a job is running, you might see one or multiple instances of BxlServer.

In effect, BxlServer is a companion to a language runtime environment that works with SQL Server to transfer data and manage tasks. BXL stands for Binary Exchange language

and refers to the data format used to move data efficiently between SQL Server and external processes.

SQL Satellite is an extensibility API, included in the database engine, that supports external code or external runtimes implemented using C or C++.

BxlServer uses SQL Satellite for these tasks:

- Reading input data
- Writing output data
- Getting input arguments
- Writing output arguments
- Error handling
- Writing STDOUT and STDERR back to client

SQL Satellite uses a custom data format that is optimized for fast data transfer between SQL Server and external script languages. It performs type conversions and defines the schemas of the input and output datasets during communications between SQL Server and the external script runtime.

The SQL Satellite can be monitored by using Windows extended events (xEvents). For more information, see [Extended Events for SQL Server Machine Learning Services](#).

Communication channels between components

Communication protocols among components and data platforms are described in this section.

- **TCP/IP**

By default, internal communications between SQL Server and the SQL Satellite use TCP/IP.

- **Named Pipes**

Internal data transport between the BxlServer and SQL Server through SQL Satellite uses a proprietary, compressed data format to enhance performance. Data is exchanged between language run times and BxlServer in BXL format, using Named Pipes.

- **ODBC**

Communications between external data science clients and a remote SQL Server instance use ODBC. The account that sends the script jobs to SQL Server must have

both permissions to connect to the instance and to run external scripts.

Additionally, depending on the task, the account might need these permissions:

- Read data used by the job
- Write data to tables: for example, when saving results to a table
- Create database objects: for example, if saving external script as part of a new stored procedure.

When SQL Server is used as the compute context for script executed from a remote client, and the executable must retrieve data from an external source, ODBC is used for writeback. SQL Server maps the identity of the user issuing the remote command to the identity of the user on the current instance, and runs the ODBC command using that user's credentials. The connection string needed to perform this ODBC call is obtained from the client code.

- **RODBC (R only)**

Additional ODBC calls can be made inside the script by using **RODBC**. RODBC is a popular R package used to access data in relational databases; however, its performance is generally slower than comparable providers used by SQL Server. Many R scripts use embedded calls to RODBC as a way of retrieving "secondary" datasets for use in analysis. For example, the stored procedure that trains a model might define a SQL query to get the data for training a model, but use an embedded RODBC call to get additional factors, to perform lookups, or to get new data from external sources such as text files or Excel.

The following code illustrates an RODBC call embedded in an R script:

```
R

library(RODBC);
connStr <- paste("Driver=SQL Server;Server=", instance_name,
";Database=", database_name, ";Trusted_Connection=true;", sep="");
dbhandle <- odbcDriverConnect(connStr)
OutputDataSet <- sqlQuery(dbhandle, "select * from table_name");
```

- **Other protocols**

Processes that might need to work in "chunks" or transfer data back to a remote client can also use the [XDF file format](#). Actual data transfer is via encoded blobs.

See Also

- [R extension in SQL Server](#)

- [Python extension in SQL Server](#)

Python language extension in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later


This article describes the Python extension for running external Python scripts with [SQL Server Machine Learning Services](#). The extension adds:

- A Python execution environment
- Anaconda distribution with the Python 3.5 runtime and interpreter
- Standard libraries and tools
- Microsoft Python packages:
 - [revoscalepy](#) for analytics at scale.
 - [microsoftml](#) for machine learning algorithms.

Installation of the Python 3.5 runtime and interpreter ensures near-complete compatibility with standard Python solutions. Python runs in a separate process from SQL Server, to guarantee that database operations are not compromised.

Python components

SQL Server includes both open-source and proprietary packages. The Python runtime installed by Setup is Anaconda 4.2 with Python 3.5. The Python runtime is installed independently of SQL tools, and is executed outside of core engine processes, in the extensibility framework. As part of the installation of Machine Learning Services with Python, you must consent to the terms of the GNU Public License.

SQL Server does not modify the Python executables, but you must use the version of Python installed by Setup because that version is the one that the proprietary packages are built and tested on. For a list of packages supported by the Anaconda distribution, see the Continuum analytics site: [Anaconda package list](#) .

The Anaconda distribution associated with a specific database engine instance can be found in the folder associated with the instance. For example, if you installed SQL Server 2017 database engine with Machine Learning Services and Python on the default instance, look under `C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES`.

Python packages added by Microsoft for parallel and distributed workloads include the following libraries.

Library	Description
revoscalepy	Supports data source objects and data exploration, manipulation, transformation, and visualization. It supports creation of remote compute contexts, as well as a various scalable machine learning models, such as rxLinMod . For more information, see revoscalepy module with SQL Server .
microsoftml	Contains machine learning algorithms that have been optimized for speed and accuracy, as well as in-line transformations for working with text and images. For more information, see microsoftml module with SQL Server .

Microsoftml and revoscalepy are tightly coupled; data sources used in microsoftml are defined as revoscalepy objects. Compute context limitations in revoscalepy transfer to microsoftml. Namely, all functionality is available for local operations, but switching to a remote compute context requires RxInSqlServer.

Using Python in SQL Server

You import the **revoscalepy** module into your Python code, and then call functions from the module, like any other Python functions.

Supported data sources include ODBC databases, SQL Server, and XDF file format to exchange data with other sources, or with R solutions. Input data for Python must be tabular. All Python results must be returned in the form of a **pandas** data frame.

Supported compute contexts include local, or remote SQL Server compute context. A remote compute context refers to code execution that starts on one computer such as a workstation, but then switches script execution to a remote computer. Switching the compute context requires that both systems have the same revoscalepy library.

Local compute context, as you might expect, includes execution of Python code on the same server as the database engine instance, with code inside T-SQL or embedded in a stored procedure. You can also run the code from a local Python IDE and have the script execute on the SQL Server computer, by defining a remote compute context.

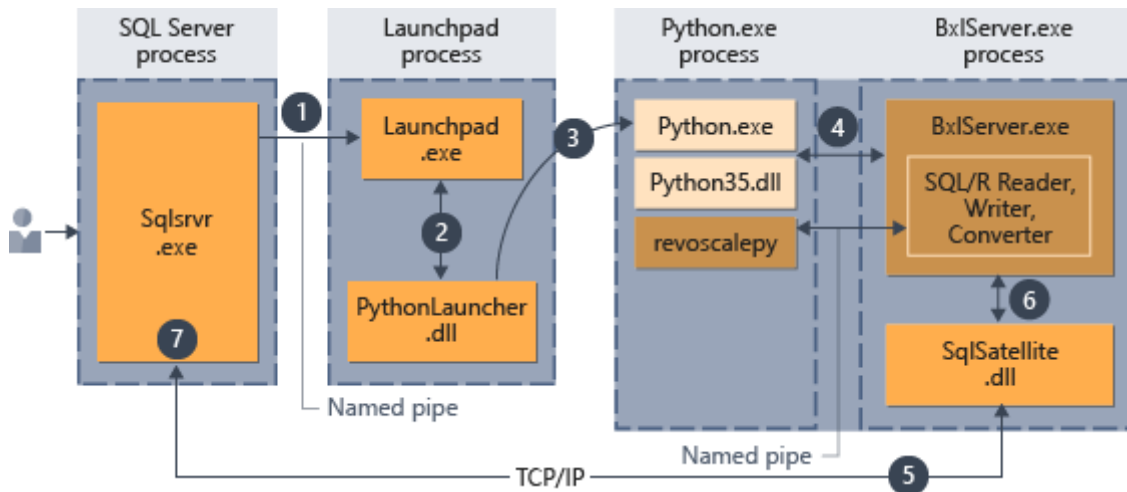
Execution architecture

The following diagrams depict the interaction of SQL Server components with the Python runtime in each of the supported scenarios: running script in-database, and remote execution from a Python terminal, using a SQL Server compute context.

Python scripts executed in-database

When you run Python "inside" SQL Server, you must encapsulate the Python script inside a special stored procedure, [sp_execute_external_script](#).

After the script has been embedded in the stored procedure, any application that can make a stored procedure call can initiate execution of the Python code. Thereafter SQL Server manages code execution as summarized in the following diagram.



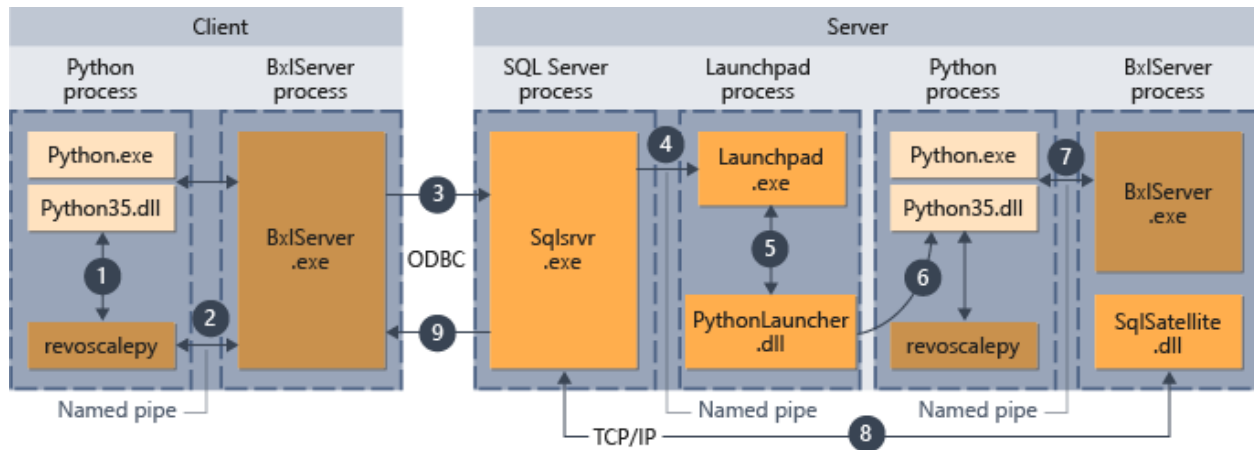
1. A request for the Python runtime is indicated by the parameter `@language='Python'` passed to the stored procedure. SQL Server sends this request to the launchpad service. In Linux, SQL uses a **launchpadd** service to communicate with a separate launchpad process for each user. See the [Extensibility architecture diagram](#) for details.
2. The launchpad service starts the appropriate launcher; in this case, PythonLauncher.
3. PythonLauncher starts the external Python35 process.
4. BxlServer coordinates with the Python runtime to manage exchanges of data, and storage of working results.
5. SQL Satellite manages communications about related tasks and processes with SQL Server.
6. BxlServer uses SQL Satellite to communicate status and results to SQL Server.
7. SQL Server gets results and closes related tasks and processes.

Python scripts executed from a remote client

You can run Python scripts from a remote computer, such as a laptop, and have them execute in the context of the SQL Server computer, if these conditions are met:

- You design the scripts appropriately
- The remote computer has installed the extensibility libraries that are used by Machine Learning Services. The [revoscalepy](#) package is required to use remote compute contexts.

The following diagram summarizes the overall workflow when scripts are sent from a remote computer.



1. For functions that are supported in **revoscalepy**, the Python runtime calls a linking function, which in turn calls BxlServer.
2. BxlServer is included with Machine Learning Services (In-Database) and runs in a separate process from the Python runtime.
3. BxlServer determines the connection target and initiates a connection using ODBC, passing credentials supplied as part of the connection string in the Python script.
4. BxlServer opens a connection to the SQL Server instance.
5. When an external script runtime is called, the launchpad service is invoked, which in turn starts the appropriate launcher: in this case, PythonLauncher.dll. Thereafter, processing of Python code is handled in a workflow similar to that when Python code is invoked from a stored procedure in T-SQL.
6. PythonLauncher makes a call to the instance of the Python that is installed on the SQL Server computer.
7. Results are returned to BxlServer.
8. SQL Satellite manages communication with SQL Server and cleanup of related job objects.
9. SQL Server passes results back to the client.

Next steps

- [revoscalepy module in SQL Server](#)
- [revoscalepy function reference](#)
- [Extensibility framework in SQL Server](#)
- [R and machine learning extensions in SQL Server](#)
- [Get Python package information](#)
- [Install Python packages with sqlmlutils](#)

R language extension in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This article describes the R extension for running external Python scripts with [SQL Server Machine Learning Services](#) and [SQL Server 2016 R Services](#). The extension adds:

- An R execution environment
- Base R distribution with standard libraries and tools
- Microsoft R libraries:
 - [RevoScaleR](#) for analytics at scale
 - [MicrosoftML](#) for machine learning algorithms. **Applies only to SQL Server 2016, SQL Server 2017, and SQL Server 2019.**
 - Other libraries for accessing data or R code in SQL Server

R components

SQL Server includes both open-source and proprietary packages. The base R libraries are installed through Microsoft's distribution of open-source R: Microsoft R Open (MRO). Current users of R should be able to port their R code and execute it as an external process on SQL Server with few or no modifications. MRO is installed independently of SQL tools, and is executed outside of core engine processes, in the extensibility framework. During installation, you must consent to the terms of the open-source license. Thereafter, you can run standard R packages without further modification just as you would in any other open-source distribution of R.

For SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x), SQL Server does not modify the base R executables, but you must use the version of R installed by Setup because that version is the one that the proprietary packages are built and tested on. For more information about how MRO differs from a base distribution of R that you might get from CRAN, see [Interoperability with R language and Microsoft R products and features](#).

The R base package distribution installed by Setup can be found in the folder associated with the instance. For example, if you installed R Services on a SQL Server default instance, the R libraries are located in this folder by default: `C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\library`. Similarly, the R tools associated

with the default instance would be located in this folder by default: `C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\bin`.

R packages added by Microsoft for parallel and distributed workloads include the following libraries.

Library	Description
RevoScaleR	Supports data source objects and data exploration, manipulation, transformation, and visualization. It supports creation of remote compute contexts, as well as a various scalable machine learning models, such as <code>rxLinMod</code> . The APIs have been optimized to analyze data sets that are too big to fit in memory and to perform computations distributed over several cores or processors. The RevoScaleR package also supports the XDF file format for faster movement and storage of data used for analysis. The XDF format uses columnar storage, is portable, and can be used to load and then manipulate data from various sources, including text, SPSS, or an ODBC connection.
MicrosoftML	Contains machine learning algorithms that have been optimized for speed and accuracy, as well as in-line transformations for working with text and images. For more information, see MicrosoftML in SQL Server . Applies only to SQL Server 2016, SQL Server 2017, and SQL Server 2019.

Using R in SQL Server

You can script R using base functions, but to benefit from multi-processing, you must import the **RevoScaleR** and **MicrosoftML** modules into your R code, and then call its functions to create models that execute in parallel.

Supported data sources include ODBC databases, SQL Server, and XDF file format to exchange data with other sources, or with R solutions. Input data must be tabular. All R results must be returned in the form of a data frame.

Supported compute contexts include local, or remote SQL Server compute context. A remote compute context refers to code execution that starts on one computer such as a workstation, but then switches script execution to a remote computer. Switching the compute context requires that both systems have the same RevoScaleR library.

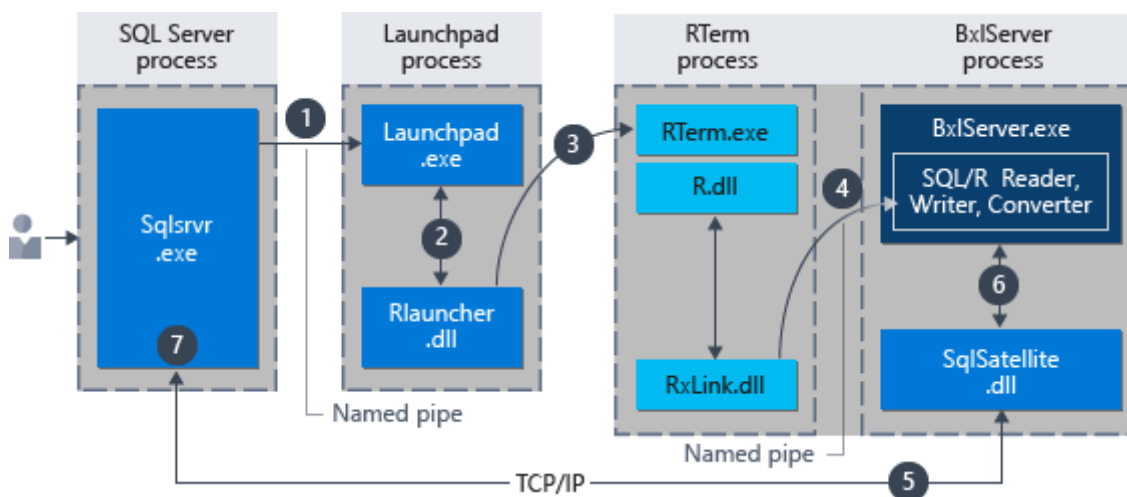
Local compute context, as you might expect, includes execution of R code on the same server as the database engine instance, with code inside T-SQL or embedded in a stored procedure. You can also run the code from a local R IDE and have the script execute on the SQL Server computer, by defining a remote compute context.

Execution architecture

The following diagrams depict the interaction of SQL Server components with the R runtime in each of the supported scenarios: running script in-database, and remote execution from an R command line, using a SQL Server compute context.

R scripts executed from SQL Server in-database

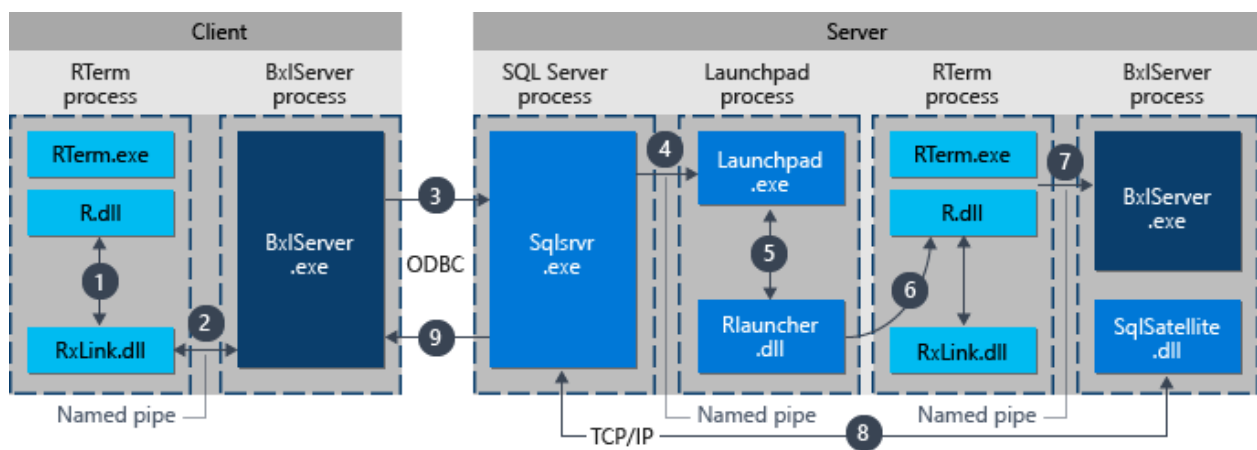
R code that is run from "inside" SQL Server is executed by calling a stored procedure. Thus, any application that can make a stored procedure call can initiate execution of R code. Thereafter SQL Server manages the execution of R code as summarized in the following diagram.



1. A request for the R runtime is indicated by the parameter `@language='R'` passed to the stored procedure, `sp_execute_external_script`. SQL Server sends this request to the launchpad service. In Linux, SQL uses a `launchpadd` service to communicate with a separate launchpad process for each user. See the [Extensibility architecture diagram](#) for details.
2. The launchpad service starts the appropriate launcher; in this case, `RLauncher`.
3. `RLauncher` starts the external R process.
4. `BxlServer` coordinates with the R runtime to manage exchanges of data with SQL Server and storage of working results.
5. SQL Satellite manages communications about related tasks and processes with SQL Server.
6. `BxlServer` uses SQL Satellite to communicate status and results to SQL Server.
7. SQL Server gets results and closes related tasks and processes.

R scripts executed from a remote client

When connecting from a remote data science client that supports Microsoft R, you can run R functions in the context of SQL Server by using the `RevoScaleR` functions. This is a different workflow from the previous one, and is summarized in the following diagram.



1. For RevoScaleR functions, the R runtime calls a linking function which in turn calls BxlServer.
2. BxlServer is provided with Microsoft R and runs in a separate process from the R runtime.
3. BxlServer determines the connection target and initiates a connection using ODBC, passing credentials supplied as part of the connection string in the R data source object.
4. BxlServer opens a connection to the SQL Server instance.
5. For an R call, the launchpad service is invoked, which in turn starts the appropriate launcher, Rlauncher. Thereafter, processing of R code is similar to the process for running R code from T-SQL.
6. Rlauncher makes a call to the instance of the R runtime that is installed on the SQL Server computer.
7. Results are returned to BxlServer.
8. SQL Satellite manages communication with SQL Server and cleanup of related job objects.
9. SQL Server passes results back to the client.

See also

- [Extensibility framework in SQL Server](#)
- [Python and machine learning extensions in SQL Server](#)

Security architecture for the extensibility framework in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This article describes the security architecture that is used to integrate the SQL Server database engine and related components with the extensibility framework in [SQL Server Machine Learning Services](#). It examines the securables, services, process identity, and permissions. Key points covered in this article include the purpose of launchpad, SQLRUserGroup and worker accounts, process isolation of external scripts, and how user identities are mapped to worker accounts.

For more information about the key concepts and components of extensibility in SQL Server, see [Extensibility architecture in SQL Server Machine Learning Services](#).

Securables for external script

An external script is submitted as an input parameter to a [system stored procedure](#) created for this purpose, or is wrapped in a stored procedure that you define. The script may be written in R, Python, or external languages such as Java or .NET. Alternatively, you might have models that are pretrained and stored in a binary format in a database table, callable in a T-SQL [PREDICT](#) function.

As the script is provided through existing database schema objects, stored procedures and tables, there are no new [securables](#) for SQL Server Machine Learning Services.

Regardless of how you are using script or, what they consist of, database objects will be created and probably saved, but no new object type is introduced for storing script. As a result, the ability to consume, create, and save database objects depends largely on database permissions already defined for your users.

Permissions

SQL Server's data security model of database logins and roles extends to external script. A SQL Server login or Windows user account is required to run external scripts that use SQL Server data or that run with SQL Server as the compute context. Database users having permissions to execute a query can access the same data from external script.

The login or user account identifies the *security principal*, who might need multiple levels of access, depending on the external script requirements:

- Permission to access the database where external scripts are enabled.
- Permissions to read data from secured objects such as tables.
- The ability to write new data to a table, such as a model, or scoring results.
- The ability to create new objects, such as tables, stored procedures that use the external script, or custom functions that use external script job.
- The right to install new packages on the SQL Server computer, or use packages provided to a group of users.

Each person who runs an external script using SQL Server as the execution context must be mapped to a user in the database. Rather than individually set database user permissions, you could create roles to manage sets of permissions, and assign users to those roles, rather than individually set user permissions.

For more information, see [Give users permission to SQL Server Machine Learning Services](#).

Permissions when using an external client tool

Users who are using script in an external client tool must have their login or account mapped to a user in the database if they need to run an external script in-database, or access database objects and data. The same permissions are required whether the external script is sent from a remote data science client or run using a T-SQL stored procedure.

For example, assume that you created an external script that runs on your local computer, and you want to run that script on SQL Server. You must ensure that the following conditions are met:

- The database allows remote connections.
- The SQL login or Windows account that you used for database access has been added to the SQL Server at the instance level.
- The SQL login or Windows user must have the permission to execute external scripts. Generally, this permission can only be added by a database administrator.
- The SQL login or Window user must be added as a user, with appropriate permissions, in each database where the external script performs any of these operations:
 - Retrieving data.
 - Writing or updating data.
 - Creating new objects, such as tables or stored procedures.

After the login or Windows user account has been provisioned and given the necessary permissions, you can run an external script on SQL Server by using a data source object in R or the **revoscalepy** library in Python, or by calling a stored procedure that contains the external script.

Whenever an external script is launched from SQL Server, the database engine security gets the security context of the user who started the job, and manages the mappings of the user or login to securable objects.

Therefore, all external scripts that are initiated from a remote client must specify the login or user information as part of the connection string.

Services used in external processing (launchpad)

The extensibility framework adds one new NT service to the [list of services](#) in a SQL Server installation: **SQL Server Launchpad (MSSQLSERVER)**.

The database engine uses the SQL Server **launchpad** service to instantiate an external script session as a separate process. The process runs under a low-privilege account. This account is distinct from SQL Server, launchpad itself, and the user identity under which the stored procedure or host query was executed. Running script in a separate process, under a low-privilege account, is the basis of the security and isolation model for external scripts in SQL Server.

SQL Server also maintains a mapping of the identity of the calling user to the low-privilege worker account used to start the satellite process. In some scenarios, where script or code calls back to SQL Server for data and operations, SQL Server is able to manage identity transfer seamlessly. Script containing SELECT statements or calling functions and other programming objects will typically succeed if the calling user has sufficient permissions.

ⓘ Note

By default, SQL Server Launchpad is configured to run under NT **Service\MSSQLLaunchpad**, which is provisioned with all necessary permissions to run external scripts. For more information about configurable options, see [SQL Server launchpad service configuration](#).

Identities used in processing (SQLRUserGroup)

SQLRUserGroup (SQL restricted user group) is created by SQL Server Setup and contains a pool of low-privilege local Windows user accounts. When an external process is needed, launchpad takes an available worker account and uses it to run a process. More specifically, launchpad activates an available worker account, maps it to the identity of the calling user, and runs the script under the worker account.

- **SQLRUserGroup** is linked to a specific instance. A separate pool of worker accounts is needed for each instance on which machine learning has been enabled. Accounts cannot be shared between instances.
- The size of the user account pool is static and the default value is 20, which supports 20 concurrent sessions. The number of external runtime sessions that can be launched simultaneously is limited by the size of this user account pool.
- Worker account names in the pool are of the format `SQLInstanceNamenn`. For example, on a default instance, **SQLRUserGroup** contains accounts named `MSSQLSERVER01`, `MSSQLSERVER02`, and so forth, on up to `MSSQLSERVER20`.

Parallelized tasks do not consume additional accounts. For example, if a user runs a scoring task that uses parallel processing, the same worker account is reused for all threads. If you intend to make heavy use of machine learning, you can increase the number of accounts used to run external scripts. For more information, see [Scale concurrent execution of external scripts in SQL Server Machine Learning Services](#).

Permissions granted to SQLRUserGroup

By default, members of **SQLRUserGroup** have read and execute permissions on files in the SQL Server `Binn`, `R_SERVICES`, and `PYTHON_SERVICES` directories. This includes access to executables, libraries, and built-in datasets in the R and Python distributions installed with SQL Server.

To protect sensitive resources on SQL Server, you can optionally define an access control list (ACL) that denies access to **SQLRUserGroup**. Conversely, you could also grant permissions to local data resources that exist on host computer, apart from SQL Server itself.

By design, **SQLRUserGroup** does not have a database login or permissions to any data. Under certain circumstances, you might want to create a login to allow loopback connections, particularly when a trusted Windows identity is the calling user. This capability is called *implied authentication*. For more information, see [Add SQLRUserGroup as a database user](#).

Identity mapping

When a session is started, launchpad maps the identity of the calling user to a worker account. The mapping of an external Windows user or valid SQL login to a worker account is valid only for the lifetime of the SQL stored procedure that executes the external script. Parallel queries from the same login are mapped to the same user worker account.

During execution, launchpad creates temporary folders to store session data, deleting them when the session concludes. The directories are access-restricted. For R, RLauncher performs this task. For Python, PythonLauncher performs this task. Each individual worker account is restricted to its own folder, and cannot access files in folders above its own level. However, the worker account can read, write, or delete children under the session working folder that was created. If you are an administrator on the computer, you can view the directories created for each process. Each directory is identified by its session GUID.

Implied authentication (loopback requests)

Implied authentication describes connection request behavior under which external processes running as low-privilege worker accounts are presented as a trusted user identity to SQL Server on loopback requests for data or operations. As a concept, implied authentication is unique to Windows authentication, in SQL Server connection strings specifying a trusted connection, on requests originating from external processes such as R or Python script. It is sometimes also referred to as a *loopback*.

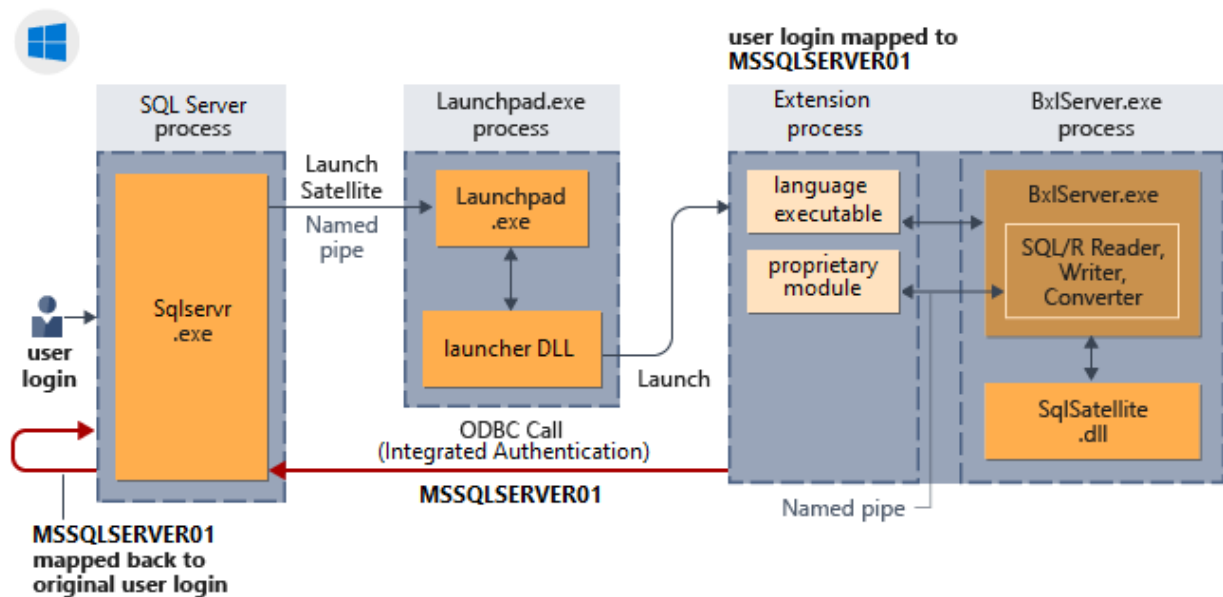
Trusted connections are workable from external script, but only with additional configuration. In the extensibility architecture, external processes run under worker accounts, inheriting permissions from the parent **SQLRUserGroup**. When a connection string specifies `Trusted_Connection=True`, the identity of the worker account is presented on the connection request, which is unknown by default to SQL Server.

To make trusted connections successful, you must create a database login for the **SQLRUserGroup**. After doing so, any trusted connection from any member of **SQLRUserGroup** has login rights to SQL Server. For step-by-step instructions, see [Add SQLRUserGroup to a database login](#).

Trusted connections are not the most widely used formulation of a connection request. When external script specifies a connection, it can be more common to use a SQL login, or a fully specified user name and password if the connection is to an ODBC data source.

How implied authentication works for external script sessions

The following diagram shows the interaction of SQL Server components with the language runtime and how it does implied authentication in Windows.



No support for Transparent Data Encryption at rest

[Transparent Data Encryption \(TDE\)](#) is not supported for data sent to or received from the external script runtime. The reason is that the external process runs outside the SQL Server process. Therefore, data used by the external runtime is not protected by the encryption features of the database engine. This behavior is no different than any other client running on the SQL Server computer that reads data from the database and makes a copy.

As a consequence, TDE is **not** applied to any data that you use in external scripts, or to any data saved to disk, or to any persisted intermediate results. However, other types of encryption, such as Windows BitLocker encryption or third-party encryption applied at the file or folder level, still apply.

In the case of [Always Encrypted](#), external runtimes do not have access to the encryption keys. Therefore, data cannot be sent to the scripts.

Next steps

In this article, you learned the components and interaction model of the security architecture built into the [extensibility framework](#). Key points covered in this article include the purpose of launchpad, SQLRUserGroup and worker accounts, process isolation of external scripts, and how user identities are mapped to worker accounts.

As a next step, review the instructions for [granting permissions](#). For servers that use Windows authentication, you should also review [Add SQLRUserGroup to a database login](#) to learn when additional configuration is required.

Using data from OLAP cubes in R

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions


The **olapR** package is an R package in [SQL Server Machine Learning Services](#) that lets you run MDX queries to get data from OLAP cubes. With this package, you don't need to create linked servers or clean up flattened rowsets; you can get OLAP data directly from R.

This article describes the API, along with an overview of OLAP and MDX for R users who might be new to multidimensional cube databases.

Important

An instance of Analysis Services can support either conventional multidimensional cubes, or tabular models, but an instance cannot support both types of models. Therefore, before you try to build an MDX query against a cube, verify that the Analysis Services instance contains multidimensional models.

What is an OLAP cube?

OLAP is short for Online Analytical Processing. OLAP solutions are widely used for capturing and storing critical business data over time. OLAP data is consumed for business analytics by a variety of tools, dashboards, and visualizations. For more information, see [Online analytical processing](#) .

Microsoft provides [Analysis Services](#), which lets you design, deploy, and query OLAP data in the form of *cubes* or *tabular models*. A cube is a multi-dimensional database. *Dimensions* are like facets of the data, or factors in R: you use dimensions to identify some particular subset of data that you want to summarize or analyze. For example, time is an important dimension, so much so that many OLAP solutions include multiple calendars defined by default, to use when slicing and summarizing data.

For performance reasons, an OLAP database often calculates summaries (or *aggregations*) in advance, and then stores them for faster retrieval. Summaries are based on *measures*, which represent formulas that can be applied to numerical data. You use the dimensions to define a subset of data, and then compute the measure over that data. For example, you would use a measure to compute the total sales for a certain

product line over multiple quarters minus taxes, to report the average shipping costs for a particular supplier, year-to-date cumulative wages paid, and so forth.

MDX, short for multidimensional expressions, is the language used for querying cubes. An MDX query typically contains a data definition that includes one or more dimensions, and at least one measure, though MDX queries can get considerably more complex, and include rolling windows, cumulative averages, sums, ranks, or percentiles.

Here are some other terms that might be helpful when you start building MDX queries:

- *Slicing* takes a subset of the cube by using values from a single dimension.
- *Dicing* creates a subcube by specifying a range of values on multiple dimensions.
- *Drill-down* navigates from a summary to details.
- *Drill-up* moves from details to a higher level of aggregation.
- *Roll-up* summarizes the data on a dimension.
- *Pivot* rotate the cube or the data selection.

How to use olapR to create MDX queries

The following article provides detailed examples of the syntax for creating or executing queries against a cube:

- [How to create MDX queries using R](#)

olapR API

The **olapR** package supports two methods of creating MDX queries:

- **Use the MDX builder.** Use the R functions in the package to generate a simple MDX query, by choosing a cube, and then setting axes and slicers. This is an easy way to build a valid MDX query if you do not have access to traditional OLAP tools, or don't have deep knowledge of the MDX language.

Not all MDX queries can be created by using this method, because MDX can be complex. However, this API supports most of the most common and useful operations, including slice, dice, drilldown, rollup, and pivot in N dimensions.

- **Copy-paste well-formed MDX.** Manually create and then paste in any MDX query. This option is the best if you have existing MDX queries that you want to reuse, or

if the query you want to build is too complex for **olapR** to handle.

After building your MDX using any client utility, such as SSMS or Excel, save the query string. Provide this MDX string as an argument to the *SSAS query handler* in the **olapR** package. The provider sends the query to the specified Analysis Services server, and passes back the results to R.

For examples of how to build an MDX query or run an existing MDX query, see [How to create MDX queries using R](#).

Known issues

This section lists some known issues and common questions about the **olapR** package.

Tabular model support

If you connect to an instance of Analysis Services that contains a tabular model, the `explore` function reports success with a return value of TRUE. However, tabular model objects are different from multidimensional objects, and the structure of a multidimensional database is different from that of a tabular model.

Although DAX (Data analysis Expressions) is the language typically used with tabular models, you can design valid MDX queries against a tabular model, if you are already familiar with MDX. You cannot use the **olapR** constructors to build valid MDX queries against a tabular model.

However, MDX queries are an inefficient way to retrieve data from a tabular model. If you need to get data from a tabular model for use in R, consider these methods instead:

- Enable DirectQuery on the model and add the server as a linked server in SQL Server.
- If the tabular model was built on a relational data mart, obtain the data directly from the source.

How to determine whether an instance contains tabular or multidimensional models

A single Analysis Services instance can contain only one type of model, though it can contain multiple models. The reason is that there are fundamental differences between tabular models and multidimensional models that control the way data is stored and processed. For example, tabular models are stored in memory and leverage columnstore

indexes to perform very fast calculations. In multidimensional models, data is stored on disk and aggregations are defined in advance and retrieved by using MDX queries.

If you connect to Analysis Services using a client such as SQL Server Management Studio, you can tell at a glance which model type is supported, by looking at the icon for the database.

You can also view and query the server properties to determine which type of model the instance supports. The **Server mode** property supports two values: *multidimensional* or *tabular*.

See the following article for general information about the two types of models:

- [Comparing multidimensional and tabular models](#)

See the following article for information about querying server properties:

- [OLE DB for OLAP Schema Rowsets](#)

Writeback is not supported

It is not possible to write the results of custom R calculations back to the cube.

In general, even when a cube is enabled for writeback, only limited operations are supported, and additional configuration might be required. We recommend that you use MDX for such operations.

- [Write-enabled dimensions](#)
- [Write-enabled partitions](#)
- [Set custom access to cell data](#)

Long-running MDX queries block cube processing

Although the **olapR** package performs only read operations, long-running MDX queries can create locks that prevent the cube from being processed. Always test your MDX queries in advance so that you know how much data should be returned.

If you try to connect to a cube that is locked, you might get an error that the SQL Server data warehouse cannot be reached. Suggested resolutions include enabling remote connections, checking the server or instance name, and so forth; however, consider the possibility of a prior open connection.

An SSAS administrator can prevent locking issues by identifying and terminating open sessions. A timeout property can also be applied to MDX queries at the server level to

force termination of all long-running queries.

Resources

If you are new to OLAP or to MDX queries, see these Wikipedia articles:

- [OLAP cubes](#) ↗
- [MDX queries](#) ↗

How to create MDX queries in R using olapR

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

The `olapR` in [SQL Server Machine Learning Services](#) package supports MDX queries against cubes hosted in [SQL Server Analysis Services](#). You can build a query against an existing cube, explore dimensions and other cube objects, and paste in existing MDX queries to retrieve data.

This article describes the two main uses of the `olapR` package:

- [Build an MDX query from R, using the constructors provided in the olapR package](#)
- [Execute an existing, valid MDX query using olapR and an OLAP provider](#)

The following operations are not supported:

- DAX queries against a tabular model
- Creation of new OLAP objects
- Writeback to partitions, including measures or sums

Build an MDX query from R

1. Define a connection string that specifies the OLAP data source (SSAS instance), and the MSOLAP provider.
2. Use the function `OlapConnection(connectionString)` to create a handle for the MDX query and pass the connection string.
3. Use the `Query()` constructor to instantiate a query object.
4. Use the following helper functions to provide more details about the dimensions and measures to include in the MDX query:
 - `cube()` Specify the name of the SSAS database. If connecting to a named instance, provide the machine name and instance name.
 - `columns()` Provide the names of the measures to use in the **ON COLUMNS** argument.

- `rows()` Provide the names of the measures to use in the **ON ROWS** argument.
- `slicers()` Specify a field or members to use as a slicer. A slicer is like a filter that is applied to all MDX query data.
- `axis()` Specify the name of an additional axis to use in the query.

An OLAP cube can contain up to 128 query axes. Generally, the first four axes are referred to as **Columns, Rows, Pages, and Chapters**.

If your query is relatively simple, you can use the functions `columns`, `rows`, etc. to build your query. However, you can also use the `axis()` function with a non-zero index value to build an MDX query with many qualifiers, or to add extra dimensions as qualifiers.

5. Pass the handle, and the completed MDX query, into one of the following functions, depending on the shape of the results:

- `executeMD` Returns a multi-dimensional array
- `execute2D` Returns a two-dimensional (tabular) data frame

Execute a valid MDX query from R

1. Define a connection string that specifies the OLAP data source (SSAS instance), and the MSOLAP provider.
2. Use the function `OlapConnection(connectionString)` to create a handle for the MDX query and pass the connection string.
3. Define an R variable to store the text of the MDX query.
4. Pass the handle and the variable containing the MDX query into the functions `executeMD` or `execute2D`, depending on the shape of the results.
 - `executeMD` Returns a multi-dimensional array
 - `execute2D` Returns a two-dimensional (tabular) data frame

Examples

The following examples are based on the AdventureWorks data mart and cube project, because that project is widely available, in multiple versions, including backup files that

can easily be restored to Analysis Services. If you don't have an existing cube, get a sample cube using either of these options:

- Create the cube that is used in these examples by following the Analysis Services tutorial up to Lesson 4: [Creating an OLAP cube](#)
- Download an existing cube as a backup, and restore it to an instance of Analysis Services. For example, this site provides a fully processed cube in zipped format: [Adventure Works Multidimensional Model SQL 2014](#). Extract the file, and then restore it to your SSAS instance. For more information, see [Backup and restore](#), or [Restore-ASDatabase Cmdlet](#).

1. Basic MDX with slicer

This MDX query selects the *measures* for count and amount of Internet sales count and sales amount, and places them on the Column axis. It adds a member of the SalesTerritory dimension as a *slicer*, to filter the query so that only the sales from Australia are used in calculations.

MDX

```
SELECT {[Measures].[Internet Sales Count], [Measures].[InternetSales-Sales Amount]} ON COLUMNS,
{[Product].[Product Line].[Product Line].MEMBERS} ON ROWS
FROM [Analysis Services Tutorial]
WHERE [Sales Territory].[Sales Territory Country].[Australia]
```

- On columns, you can specify multiple measures as elements of a comma-separated string.
- The Row axis uses all possible values (all MEMBERS) of the "Product Line" dimension.
- This query would return a table with three columns, containing a *rollup* summary of Internet sales from all countries/regions.
- The WHERE clause specifies the *slicer axis*. In this example, the slicer uses a member of the **SalesTerritory** dimension to filter the query so that only the sales from Australia are used in calculations.

To build this query using the functions provided in olapR

R

```
cnnstr <- "Data Source=localhost; Provider=MSOLAP; initial catalog=Analysis Services Tutorial"
```

```

ocs <- OlapConnection(cnnstr)

qry <- Query()
cube(qry) <- "[Analysis Services Tutorial]"
columns(qry) <- c("[Measures].[Internet Sales Count]", "[Measures].[Internet
Sales-Sales Amount]")
rows(qry) <- c("[Product].[Product Line].[Product Line].MEMBERS")
slicers(qry) <- c("[Sales Territory].[Sales Territory Country].[Australia]")

result1 <- executeMD(ocs, qry)

```

For a named instance, be sure to escape any characters that could be considered control characters in R. For example, the following connection string references an instance OLAP01, on a server named ContosoHQ:

R

```

cnnstr <- "Data Source=ContosoHQ\OLAP01; Provider=MSOLAP; initial
catalog=Analysis Services Tutorial"

```

To run this query as a predefined MDX string

R

```

cnnstr <- "Data Source=localhost; Provider=MSOLAP; initial catalog=Analysis
Services Tutorial"
ocs <- OlapConnection(cnnstr)

mdx <- "SELECT {[Measures].[Internet Sales Count], [Measures].
[InternetSales-Sales Amount]} ON COLUMNS, {[Product].[Product Line].[Product
Line].MEMBERS} ON ROWS FROM [Analysis Services Tutorial] WHERE [Sales
Territory].[Sales Territory Country].[Australia]"

result2 <- execute2D(ocs, mdx)

```

If you define a query by using the MDX builder in SQL Server Management Studio and then save the MDX string, it will number the axes starting at 0, as shown here:

MDX

```

SELECT {[Measures].[Internet Sales Count], [Measures].[Internet Sales-Sales
Amount]} ON AXIS(0),
    {[Product].[Product Line].[Product Line].MEMBERS} ON AXIS(1)
FROM [Analysis Services Tutorial]
WHERE [Sales Territory].[Sales Territory Country].[Australia]

```

You can still run this query as a predefined MDX string. However, to build the same query using R using the `axis()` function, you must renumber the axes starting at 1.

2. Explore cubes and their fields on an SSAS instance

You can use the `explore` function to return a list of cubes, dimensions, or members to use in constructing your query. This is handy if you don't have access to other OLAP browsing tools, or if you want to programmatically manipulate or construct the MDX query.

To list the cubes available on the specified connection

To view all cubes or perspectives on the instance that you have permission to view, provide the handle as an argument to `explore`.

ⓘ Important

The final result is **not** a cube; TRUE merely indicates that the metadata operation was successful. An error is thrown if arguments are invalid.

R

```
cnstr <- "Data Source=localhost; Provider=MSOLAP; initial catalog=Analysis  
Services Tutorial"  
ocs <- OlapConnection(cnstr)  
explore(ocs)
```

Results

Analysis Services Tutorial

Internet Sales

Reseller Sales

Sales Summary

[1] TRUE

To get a list of cube dimensions

To view all dimensions in the cube or perspective, specify the cube or perspective name.

R

```
cnstr <- "Data Source=localhost; Provider=MSOLAP; initial catalog=Analysis Services Tutorial"
ocs <- OlapConnection(cnstr)
explore(ocs, "Sales")
```

Results

Customer

Date

Region

To return all members of the specified dimension and hierarchy

After defining the source and creating the handle, specify the cube, dimension, and hierarchy to return. In the return results, items that are prefixed with -> represent children of the previous member.

R

```
cnstr <- "Data Source=localhost; Provider=MSOLAP; initial catalog=Analysis Services Tutorial"
ocs <- OlapConnection(cnstr)
explore(ocs, "Analysis Services Tutorial", "Product", "Product Categories", "Category")
```

Results

Accessories

Bikes

Clothing

Components

-> Assembly Components

-> Assembly Components

See also

[Using data from OLAP cubes in R](#)

Plot histograms in Python

Article • 08/10/2023

Applies to:  [SQL Server](#)  [Azure SQL Database](#)  [Azure SQL Managed Instance](#)

This article describes how to plot data using the Python package [pandas'.hist\(\)'](#). A SQL database is the source used to visualize the histogram data intervals that have consecutive, non-overlapping values.

Prerequisites

- [SQL Server for Windows or for Linux](#)
- Azure Data Studio. To install, see [Azure Data Studio](#).
- [Restore sample DW database](#) to get sample data used in this article.

Verify restored database

You can verify that the restored database exists by querying the `Person.CountryRegion` table:

SQL

```
USE AdventureWorksDW;  
SELECT * FROM Person.CountryRegion;
```

Install Python packages

[Download and Install Azure Data Studio](#).

Install the following Python packages:

- `pyodbc`
- `pandas`
- `sqlalchemy`
- `matplotlib`

To install these packages:

1. In your Azure Data Studio notebook, select **Manage Packages**.

2. In the **Manage Packages** pane, select the **Add new** tab.
3. For each of the following packages, enter the package name, select **Search**, then select **Install**.

Plot histogram

The distributed data displayed in the histogram is based on a SQL query from `AdventureWorksDW2022`. The histogram visualizes data and the frequency of data values.

Edit the connection string variables: 'server', 'database', 'username', and 'password' to connect to SQL Server database.

To create a new notebook:

1. In Azure Data Studio, select **File**, select **New Notebook**.
2. In the notebook, select kernel **Python3**, select the **+code**.
3. Paste code in notebook, select **Run All**.

Python

```
import pyodbc
import pandas as pd
import matplotlib
import sqlalchemy

from sqlalchemy import create_engine

matplotlib.use('TkAgg', force=True)
from matplotlib import pyplot as plt

# Some other example server values are
# server = 'localhost\\sqlexpress' # for a named instance
# server = 'myserver,port' # to specify an alternate port
server = 'servername'
database = 'AdventureWorksDW2022'
username = 'yourusername'
password = 'databasename'

url = 'mssql+pyodbc://{user}:{passwd}@{host}:{port}/{db}?
driver=SQL+Server'.format(user=username, passwd=password, host=server,
port=port, db=database)
engine = create_engine(url)

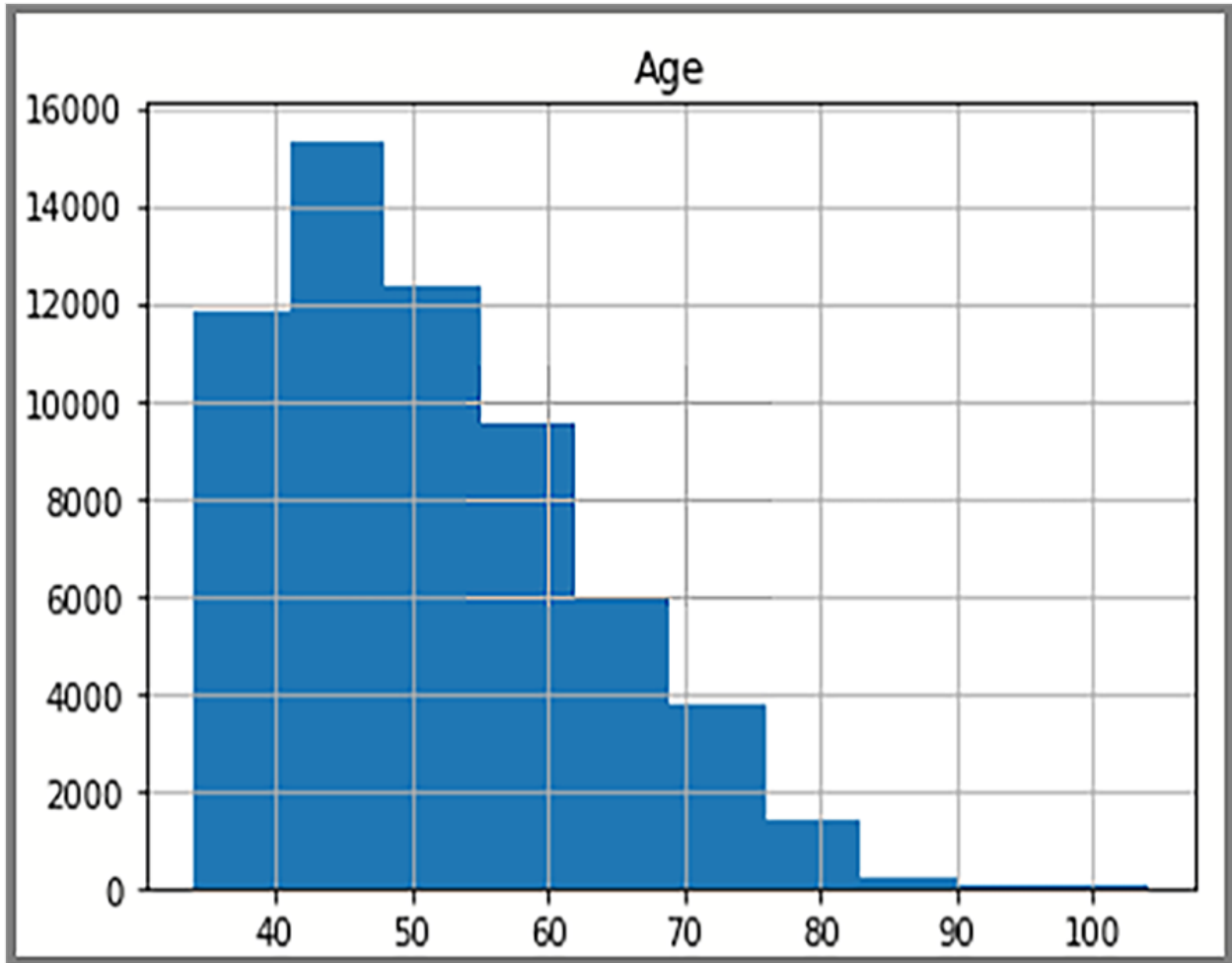
sql = "SELECT DATEDIFF(year, c.BirthDate, GETDATE()) AS Age FROM [dbo].
[FactInternetSales] s INNER JOIN dbo.DimCustomer c ON s.CustomerKey =
c.CustomerKey"

df = pd.read_sql(sql, engine)
df.hist(bins=50)
```



```
plt.show()
```

The display shows the age distribution of customers in the `FactInternetSales` table.



Insert data from a SQL table into a Python pandas dataframe

Article • 02/28/2023

Applies to:  [SQL Server](#)  [Azure SQL Database](#)  [Azure SQL Managed Instance](#)

This article describes how to insert SQL data into a [pandas](#) dataframe using the [pyodbc](#) package in Python. The rows and columns of data contained within the dataframe can be used for further data exploration.

Prerequisites

- [SQL Server for Windows](#) or [for Linux](#)
- Azure Data Studio. To install, see [Azure Data Studio](#).
- [Restore sample database](#) to get sample data used in this article.

Verify restored database

You can verify that the restored database exists by querying the `Person.CountryRegion` table:

SQL

```
USE AdventureWorks;  
SELECT * FROM Person.CountryRegion;
```

Install Python packages

[Download and Install Azure Data Studio](#).

Install the following Python packages:

- `pyodbc`
- `pandas`

To install these packages:

1. In your Azure Data Studio notebook, select **Manage Packages**.
2. In the **Manage Packages** pane, select the **Add new** tab.

3. For each of the following packages, enter the package name, click **Search**, then click **Install**.

Insert data

Use the following script to select data from Person.CountryRegion table and insert into a dataframe. Edit the connection string variables: 'server', 'database', 'username', and 'password' to connect to SQL.

To create a new notebook:

1. In Azure Data Studio, select **File**, select **New Notebook**.
2. In the notebook, select kernel **Python3**, select the **+code**.
3. Paste code in notebook, select **Run All**.

Python

```
import pyodbc
import pandas as pd
# Some other example server values are
# server = 'localhost\\sqlexpress' # for a named instance
# server = 'myserver,port' # to specify an alternate port
server = 'servername'
database = 'AdventureWorks'
username = 'yourusername'
password = 'databasename'
cnxn = pyodbc.connect('DRIVER={SQL
Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+
password)
cursor = cnxn.cursor()
# select 26 rows from SQL table to insert in dataframe.
query = "SELECT [CountryRegionCode], [Name] FROM Person.CountryRegion;"
df = pd.read_sql(query, cnxn)
print(df.head(26))
```

Output

The `print` command in the preceding script displays the rows of data from the `pandas` dataframe `df`.

text

	CountryRegionCode	Name
0	AF	Afghanistan
1	AL	Albania
2	DZ	Algeria
3	AS	American Samoa
4	AD	Andorra

5	AO	Angola
6	AI	Anguilla
7	AQ	Antarctica
8	AG	Antigua and Barbuda
9	AR	Argentina
10	AM	Armenia
11	AW	Aruba
12	AU	Australia
13	AT	Austria
14	AZ	Azerbaijan
15	BS	Bahamas, The
16	BH	Bahrain
17	BD	Bangladesh
18	BB	Barbados
19	BY	Belarus
20	BE	Belgium
21	BZ	Belize
22	BJ	Benin
23	BM	Bermuda
24	BT	Bhutan
25	BO	Bolivia

Next steps

- [Insert Python dataframe into SQL](#)

Insert Python dataframe into SQL table

Article • 02/28/2023

Applies to:  [SQL Server](#)  [Azure SQL Database](#)  [Azure SQL Managed Instance](#)

This article describes how to insert a [pandas](#) dataframe into a SQL database using the [pyodbc](#) package in Python.

Prerequisites

- [SQL Server for Windows or for Linux](#)
- Azure Data Studio. To install, see [Download and install Azure Data Studio](#).
- Follow the steps in [AdventureWorks sample databases](#) to restore the OLTP version of the AdventureWorks sample database for your version of SQL Server.

You can verify that the database was restored correctly by querying the **HumanResources.Department** table:

SQL

```
USE AdventureWorks ;  
SELECT * FROM HumanResources.Department ;
```

Install Python packages

1. In Azure Data Studio, open a new notebook and connect to the Python 3 kernel.
2. Select **Manage Packages**.



3. In the **Manage Packages** pane, select the **Add new** tab.
4. For each of the following packages, enter the package name, click **Search**, then click **Install**.
 - pyodbc
 - pandas

Create a sample CSV file

Copy the following text and save it to a file named `department.csv`.

text

```
DepartmentID,Name,GroupName,
1,Engineering,Research and Development,
2,Tool Design,Research and Development,
3,Sales,Sales and Marketing,
4,Marketing,Sales and Marketing,
5,Purchasing,Inventory Management,
6,Research and Development,Research and Development,
7,Production,Manufacturing,
8,Production Control,Manufacturing,
9,Human Resources,Executive General and Administration,
10,Finance,Executive General and Administration,
11,Information Services,Executive General and Administration,
12,Document Control,Quality Assurance,
13,Quality Assurance,Quality Assurance,
14,Facilities and Maintenance,Executive General and Administration,
15,Shipping and Receiving,Inventory Management,
16,Executive,Executive General and Administration
```

Create a new database table

1. Follow the steps in [Connect to a SQL Server](#) to connect to the AdventureWorks database.
2. Create a table named `HumanResources.DepartmentTest`. The SQL table will be used for the dataframe insertion.

SQL

```
CREATE TABLE [HumanResources].[DepartmentTest](
  [DepartmentID] [smallint] NOT NULL,
  [Name] [dbo].[Name] NOT NULL,
  [GroupName] [dbo].[Name] NOT NULL
)
GO
```

Load a dataframe from the CSV file

Use the Python `pandas` package to create a dataframe, load the CSV file, and then load the dataframe into the new SQL table, `HumanResources.DepartmentTest`.

1. Connect to the **Python 3** kernel.
2. Paste the following code into a code cell, updating the code with the correct values for `server`, `database`, `username`, `password`, and the location of the CSV file.

```
Python

import pyodbc
import pandas as pd
# insert data from csv file into dataframe.
# working directory for csv file: type "pwd" in Azure Data Studio or Linux
# working directory in Windows c:\users\username
df = pd.read_csv("c:\\user\\username\\department.csv")
# Some other example server values are
# server = 'localhost\\sqlexpress' # for a named instance
# server = 'myserver,port' # to specify an alternate port
server = 'yourservername'
database = 'AdventureWorks'
username = 'username'
password = 'yourpassword'
cnxn = pyodbc.connect('DRIVER={SQL
Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+
password)
cursor = cnxn.cursor()
# Insert Dataframe into SQL Server:
for index, row in df.iterrows():
    cursor.execute("INSERT INTO HumanResources.DepartmentTest
(DepartmentID,Name,GroupName) values(?,?,?)", row.DepartmentID,
row.Name, row.GroupName)
cnxn.commit()
cursor.close()
```

3. Run the cell.

Confirm data in the database

Connect to the SQL kernel and AdventureWorks database and run the following SQL statement to confirm the table was successfully loaded with data from the dataframe.

```
SQL

SELECT count(*) from HumanResources.DepartmentTest;
```

Results

```
Bash
```

(No column name)

16

Next steps

- [Plot a histogram for data exploration with Python](#)

Use ODBC to save and load R objects in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Learn how to use the **RevoScaleR** package to store serialized R objects in a table and then load the object from the table as needed with [SQL Server Machine Learning Services](#). This can be used when training and saving a model, and then use it later for scoring or analysis.

RevoScaleR package

The **RevoScaleR** package includes serialization and deserialization functions that can R objects compactly to SQL Server and then read the objects from the table. In general, each function call uses a simple key value store, in which the key is the name of the object, and the value associated with the key is the varbinary R object to be moved in or out of a table.

To save R objects to SQL Server directly from an R environment, you must:

- established a connection to SQL Server using the *RxOdbcData* data source.
- Call the new functions over the ODBC connection
- Optionally, you can specify that the object not be serialized. Then, choose a new compression algorithm to use instead of the default compression algorithm.

By default, any object that you call from R to move to SQL Server is serialized and compressed. Conversely, when you load an object from a SQL Server table to use in your R code, the object is deserialized and decompressed.

List of new functions

- `rxWriteObject` writes an R object into SQL Server using the ODBC data source.
- `rxReadObject` reads an R object from a SQL Server database, using an ODBC data source
- `rxDeleteObject` deletes an R object from the SQL Server database specified in the ODBC data source. If there are multiple objects identified by the key/version combination, all are deleted.

- `rxListKeys` lists as key-value pairs all the available objects. This helps you determine the names and versions of the R objects.

For detailed help on the syntax of each function, use R help. Details are also available in the [ScaleR reference](#).

How to store R objects in SQL Server using ODBC

This procedure demonstrates how you can use the new functions to create a model and save it to SQL Server.

1. Set up the connection string for the SQL Server.

```
R  
  
conStr <- 'Driver={SQL  
Server};Server=localhost;Database=storedb;Trusted_Connection=true'
```

2. Create an `RxOdbcData` data source object in R using the connection string.

```
R  
  
ds <- RxOdbcData(table="objects", connectionString=conStr)
```

3. Delete the table if it already exists, and you don't want to track old versions of the objects.

```
R  
  
if(rxSqlServerTableExists(ds@table, ds@connectionString)) {  
  rxSqlServerDropTable(ds@table, ds@connectionString)  
}
```

4. Define a table that can be used to store binary objects.

```
R  
  
ddl <- paste(" CREATE TABLE [", ds@table, "  
(", " [id] varchar(200) NOT NULL,  
", " [value] varbinary(max),  
", " CONSTRAINT unique_id UNIQUE (id))",  
sep = "")
```

5. Open the ODBC connection to create the table, and when the DDL statement has completed, close the connection.

R

```
rxOpen(ds, "w")
rxExecuteSQLDDL(ds, ddl)
rxClose(ds)
```

6. Generate the R objects that you want to store.

R

```
infertLogit <- rxLogit(case ~ age + parity + education + spontaneous +
  induced,
  data = infert)
```

7. Use the *RxOdbcData* object created earlier to save the model to the database.

R

```
rxWriteObject(ds, "logit.model", infertLogit)
```

How to read R objects from SQL Server using ODBC

This procedure demonstrates how you can use the new functions to load a model from SQL Server.

1. Set up the connection string for the SQL Server.

R

```
conStr2 <- 'Driver={SQL
  Server};Server=localhost;Database=storedb;Trusted_Connection=true'
```

2. Create an *RxOdbcData* data source object in R, using the connection string.

R

```
ds <- RxOdbcData(table="objects", connectionString=conStr2)
```

3. Read the model from the table by specifying its R object name.

R

```
infertLogit2 <- rxReadObject(ds, "logit.model")
```

Next steps

- [What is SQL Server Machine Learning Services?](#)

Creating multiple models using rxExecBy


Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Learn how to use the **rxExecBy** function in RevoScaleR to parallel process multiple related models with [SQL Server Machine Learning Services](#). Rather than train one large model based on data from multiple similar entities, you can quickly create many related models, each using data specific to a single entity.

What rxExecBy can do

For example, suppose you are monitoring device failures, capturing data for many different types of equipment. By using **rxExecBy**, you can provide a single large dataset as input, specify a column on which to stratify the dataset, such as device type, and then create multiple models for individual devices.

This use case has been termed "[pleasingly parallel](#)"  because it breaks a large complicated problem into component parts for concurrent processing.

Typical applications of this approach include forecasting for individual household smart meters, creating revenue projections for separate product lines, or creating models for loan approvals that are tailored to individual bank branches.

How rxExecBy works

The **rxExecBy** function in RevoScaleR is designed for high-volume parallel processing over a large number of small data sets.

1. You call the **rxExecBy** function as part of your R code, and pass a dataset of unordered data.
2. Specify the partition by which the data should be grouped and sorted.
3. Define a transformation or modeling function that should be applied to each data partition
4. When the function executes, the data queries are processed in parallel if your environment supports it. Moreover, the modeling or transformation tasks are distributed among individual cores and executed in parallel. Supported compute context for these operations include RxSpark and RxInSqlServer.
5. Multiple results are returned.

rxExecBy syntax and examples

`rxExecBy` takes four inputs, one of the inputs being a dataset or data source object that can be partitioned on a specified **key** column. The function returns an output for each partition. The form of the output depends on the function that is passed as an argument. For example, if you pass a modeling function such as `rxLinMod`, you could return a separate trained model for each partition of the dataset.

Supported functions

Modeling: `rxLinMod`, `rxLogit`, `rxGlm`, `rxDtree`

Scoring: `rxPredict`,

Transformation or analysis: `rxCovCor`

Example

The following example demonstrates how to create multiple models using the Airline dataset, which is partitioned on the [DayOfWeek] column. The user-defined function, `delayFunc`, is applied to each of the partitions by calling `rxExecBy`. The function creates separate models for Mondays, Tuesdays, and so forth.

SQL

```
EXEC sp_execute_external_script
@language = N'R'
, @script = N'
delayFunc <- function(key, data, params) {
  df <- rxImport(inData = airlineData)
  rxLinMod(ArrDelay ~ CRSDepTime, data = df)
}
OutputDataSet <- rxExecBy(airlineData, c("DayOfWeek"), delayFunc)
'
, @input_data_1 = N'select ArrDelay, DayOfWeek, CRSDepTime from
AirlineDemoSmall]'
, @input_data_1_name = N'airlineData'
```

If you get the error, `varsToPartition is invalid`, check whether the name of the key column or columns is typed correctly. The R language is case-sensitive.

This particular example is not optimized for SQL Server, and you could in many cases achieve better performance by using SQL to group the data. However, using `rxExecBy`,

you can create parallel jobs from R.

The following example illustrates the process in R, using SQL Server as the compute context:

R

```
sqlServerConnString <-  
"SERVER=hostname;DATABASE=TestDB;UID=DBUser;PWD=Password;"  
inTable <- paste("airlinedemosmall")  
sqlServerDataDS <- RxSqlServerData(table = inTable, connectionString =  
sqlServerConnString)  
  
# user function  
".Count" <- function(keys, data, params)  
{  
  myDF <- rxImport(inData = data)  
  return (nrow(myDF))  
}  
  
# Set SQL Server compute context with level of parallelism = 2  
sqlServerCC <- RxInSqlServer(connectionString = sqlServerConnString,  
numTasks = 4)  
rxSetComputeContext(sqlServerCC)  
  
# Execute rExecBy in SQL Server compute context  
sqlServerCCResults <- rExecBy(inData = sqlServerDataDS, keys =  
c("DayOfWeek"), func = .Count)
```

Next steps

- [What is SQL Server Machine Learning Services?](#)

Data type mappings between Python and SQL Server

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

This article lists the supported data types, and the data type conversions performed, when using the Python integration feature in SQL Server Machine Learning Services.

Python supports a limited number of data types in comparison to SQL Server. As a result, whenever you use data from SQL Server in Python scripts, SQL data might be implicitly converted to a compatible Python data type. However, often an exact conversion cannot be performed automatically and an error is returned.

Python and SQL Data Types

This table lists the implicit conversions that are provided. Other data types are not supported.

SQL type	Python type	Description
bigint	<code>float64</code>	
binary	<code>bytes</code>	
bit	<code>bool</code>	
char	<code>str</code>	
date	<code>datetime</code>	
datetime	<code>datetime</code>	Supported with SQL Server 2017 CU6 and above (with NumPy arrays of type <code>datetime.datetime</code> or Pandas <code>pandas.Timestamp</code>). <code>sp_execute_external_script</code> now supports <code>datetime</code> types with fractional seconds.
float	<code>float64</code>	
nchar	<code>str</code>	
nvarchar	<code>str</code>	
nvarchar(max)	<code>str</code>	

SQL type	Python type	Description
real	float64	
smalldatetime	datetime	
smallint	int32	
tinyint	int32	
uniqueidentifier	str	
varbinary	bytes	
varbinary(max)	bytes	
varchar(n)	str	
varchar(max)	str	

See also

- [Data type mappings between R and SQL Server](#)

Data type mappings between R and SQL Server

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

This article lists the supported data types, and the data type conversions performed, when using the R integration feature in SQL Server Machine Learning Services.

Base R version

SQL Server 2016 R Services and SQL Server Machine Learning Services with R are aligned with specific releases of Microsoft R Open. For example, the latest release, SQL Server 2019 Machine Learning Services, is built on Microsoft R Open 3.5.2.

To view the R version associated with a particular instance of SQL Server, open **RGui** in the SQL instance. For example, the path for the default instance in SQL Server 2019 would be: `C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\R_SERVICES\bin\x64\Rgui.exe`.

The tool loads base R and other libraries. Package version information is provided in a notification for each package that is loaded at session start up.

R and SQL Data Types

While SQL Server supports several dozen data types, R has a limited number of scalar data types (numeric, integer, complex, logical, character, date/time, and raw). As a result, whenever you use data from SQL Server in R scripts, data might be implicitly converted to a compatible data type. However, often an exact conversion cannot be performed automatically, and an error is returned, such as "Unhandled SQL data type".

This section lists the implicit conversions that are provided, and lists unsupported data types. Some guidance is provided for mapping data types between R and SQL Server.

Implicit data type conversions

The following table shows the changes in data types and values when data from SQL Server is used in an R script and then returned to SQL Server.

SQL type	R class	RESULT SET type	Comments
bigint	numeric	float	Executing an R script with <code>sp_execute_external_script</code> allows bigint data type as input data. However, because they are converted to R's numeric type, it suffers a precision loss with values that are very high or have decimal point values. R only support up to 53-bit integers and then it will start to have precision loss.
binary(n) n <= 8000	raw	varbinary(max)	Only allowed as input parameter and output
bit	logical	bit	
char(n) n <= 8000	character	varchar(max)	The input data frame (input_data_1) are created without explicitly setting of <code>stringsAsFactors</code> parameter so the column type will depend on the <code>default.stringsAsFactors()</code> in R
datetime	POSIXct	datetime	Represented as GMT
date	POSIXct	datetime	Represented as GMT
decimal(p,s)	numeric	float	Executing an R script with <code>sp_execute_external_script</code> allows decimal data type as input data. However, because they are converted to R's numeric type, it suffers a precision loss with values that are very high or have decimal point values. <code>sp_execute_external_script</code> with an R script does not support the full range of the data type and would alter the last few decimal digits especially those with fraction.
float	numeric	float	
int	integer	int	
money	numeric	float	Executing an R script with <code>sp_execute_external_script</code> allows money data type as input data. However, because they are converted to R's numeric type, it suffers a precision loss with values that are very high or have decimal point values. Sometimes cent values would be imprecise and a warning would be issued: <i>Warning: unable to precisely represent cents values.</i>

SQL type	R class	RESULT SET type	Comments
numeric(p,s)	numeric	float	Executing an R script with <code>sp_execute_external_script</code> allows numeric data type as input data. However, because they are converted to R's numeric type, it suffers a precision loss with values that are very high or have decimal point values. <code>sp_execute_external_script</code> with an R script does not support the full range of the data type and would alter the last few decimal digits especially those with fraction.
real	numeric	float	
smalldatetime	POSIXct	datetime	Represented as GMT
smallint	integer	int	
smallmoney	numeric	float	
tinyint	integer	int	
uniqueidentifier	character	varchar(max)	
varbinary(n) n <= 8000	raw	varbinary(max)	Only allowed as input parameter and output
varbinary(max)	raw	varbinary(max)	Only allowed as input parameter and output
varchar(n) n <= 8000	character	varchar(max)	The input data frame (input_data_1) are created without explicitly setting of <code>stringsAsFactors</code> parameter so the column type will depend on the <code>default.stringsAsFactors()</code> in R

Data types not supported by R

Of the categories of data types supported by the [SQL Server type system](#), the following types are likely to pose problems when passed to R code:

- Data types listed in the **Other** section of the SQL type system article: `cursor`, `timestamp`, `hierarchyid`, `uniqueidentifier`, `sql_variant`, `xml`, `table`
- All spatial types
- `image`

Data types that might convert poorly

- Most datetime types should work, except for **datetimeoffset**.
- Most numeric data types are supported, but conversions might fail for **money** and **smallmoney**.
- **varchar** is supported, but because SQL Server uses Unicode as a rule, use of **nvarchar** and other Unicode text data types is recommended where possible.
- Functions from the RevoScaleR library prefixed with rx can handle the SQL binary data types (**binary** and **varbinary**), but in most scenarios special handling will be required for these types. Most R code cannot work with binary columns.

For more information about SQL Server data types, see [Data Types \(Transact-SQL\)](#)

Changes in data types between SQL Server versions

Microsoft SQL Server 2016 and later include improvements in data type conversions and in several other operations. Most of these improvements offer increased precision when you deal with floating-point types, as well as minor changes to operations on classic **datetime** types.

These improvements are all available by default when you use a database compatibility level of 130 or later. However, if you use a different compatibility level, or connect to a database using an older version, you might see differences in the precision of numbers or other results.

For more information, see [SQL Server 2016 improvements in handling some data types and uncommon operations](#) [↗](#).

Verify R and SQL data schemas in advance

In general, whenever you have any doubt about how a particular data type or data structure is being used in R, use the `str()` function to get the internal structure and type of the R object. The result of the function is printed to the R console and is also available in the query results, in the **Messages** tab in Management Studio.

When retrieving data from a database for use in R code, you should always eliminate columns that cannot be used in R, as well as columns that are not useful for analysis, such as GUIDS (uniqueidentifier), timestamps and other columns used for auditing, or lineage information created by ETL processes.

Note that inclusion of unnecessary columns can greatly reduce the performance of R code, especially if high cardinality columns are used as factors. Therefore, we

recommend that you use SQL Server system stored procedures and information views to get the data types for a given table in advance, and eliminate or convert incompatible columns. For more information, see [Information Schema Views in Transact-SQL](#)

If a particular SQL Server data type is not supported by R, but you need to use the columns of data in the R script, we recommend that you use the [CAST and CONVERT \(Transact-SQL\)](#) functions to ensure that the data type conversions are performed as intended before using the data in your R script.

Warning

If you use the `rxDataStep` to drop incompatible columns while moving data, be aware that the arguments `varsToKeep` and `varsToDrop` are not supported for the `RxSqlServerData` data source type.

Examples

Example 1: Implicit conversion

The following example demonstrates how data is transformed when making the round-trip between SQL Server and R.

The query gets a series of values from a SQL Server table, and uses the stored procedure [sp_execute_external_script](#) to output the values using the R runtime.

SQL

```
CREATE TABLE MyTable (  
  c1 int,  
  c2 varchar(10),  
  c3 uniqueidentifier  
);  
go  
INSERT MyTable VALUES(1, 'Hello', newid());  
INSERT MyTable VALUES(-11, 'world', newid());  
SELECT * FROM MyTable;  
  
EXECUTE sp_execute_external_script  
  @language = N'R'  
  , @script = N'  
inputDataSet["cR"] <- c(4, 2)  
str(inputDataSet)  
outputDataSet <- inputDataSet'  
  , @input_data_1 = N'SELECT c1, c2, c3 FROM MyTable'  
  , @input_data_1_name = N'inputDataSet'
```

```
, @output_data_1_name = N'outputDataSet'  
WITH RESULT SETS((C1 int, C2 varchar(max), C3 varchar(max), C4 float));
```

Results

Row #	C1	C2	C3	C4
1	1	Hello	6e225611-4b58-4995-a0a5-554d19012ef1	4
2	-11	world	6732ea46-2d5d-430b-8ao1-86e7f3351c3e	2

Note the use of the `str` function in R to get the schema of the output data. This function returns the following information:

Output

```
'data.frame': 2 obs. of 4 variables:  
 $ c1: int  1 -11  
 $ c2: Factor w/ 2 levels "Hello","world": 1 2  
 $ c3: Factor w/ 2 levels "6732EA46-2D5D-430B-8A01-86E7F3351C3E",...: 2 1  
 $ cR: num  4 2
```

From this, you can see that the following data type conversions were implicitly performed as part of this query:

- **Column C1.** The column is represented as `int` in SQL Server, `integer` in R, and `int` in the output result set.

No type conversion was performed.

- **Column C2.** The column is represented as `varchar(10)` in SQL Server, `factor` in R, and `varchar(max)` in the output.

Note how the output changes; any string from R (either a factor or a regular string) will be represented as `varchar(max)`, no matter what the length of the strings is.

- **Column C3.** The column is represented as `uniqueidentifier` in SQL Server, `character` in R, and `varchar(max)` in the output.

Note the data type conversion that happens. SQL Server supports the `uniqueidentifier` but R does not; therefore, the identifiers are represented as strings.

- **Column C4.** The column contains values generated by the R script and not present in the original data.

Example 2: Dynamic column selection using R

The following example shows how you can use R code to check for invalid column types. The gets the schema of a specified table using the SQL Server system views, and removes any columns that have a specified invalid type.

R

```
connStr <- "Server=.;Database=TestDB;Trusted_Connection=Yes"
data <- RxSqlServerData(connectionString = connStr, sqlQuery = "SELECT
COLUMN_NAME FROM TestDB.INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME =
N'testdata' AND DATA_TYPE <> 'image';")
columns <- rxImport(data)
columnList <- do.call(paste, c(as.list(columns$COLUMN_NAME), sep = ","))
sqlQuery <- paste("SELECT", columnList, "FROM testdata")
```

See also

- [Data type mappings between Python and SQL Server](#)

Modify R/Python code to run in SQL Server (In-Database) instances

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

This article provides high-level guidance on how to modify R or Python code to run as a SQL Server stored procedure to improve performance when accessing SQL data.

When you move R/Python code from a local IDE or other environment to SQL Server, the code generally works without further modification. This is especially true for simple code, such as a function that takes some inputs and returns a value. It's also easier to port solutions that use the **RevoScaleR/revoscalepy** packages, which support execution in different execution contexts with minimal changes. Note that **MicrosoftML** applies to SQL Server 2016 (13.x), SQL Server 2017 (14.x), and SQL Server 2019 (15.x), and does not appear in SQL Server 2022 (16.x).

However, your code might require substantial changes if any of the following apply:

- You use libraries that access the network or that cannot be installed on SQL Server.
- The code makes separate calls to data sources outside SQL Server, such as Excel worksheets, files on shares, and other databases.
- You want to parameterize the stored procedure and run the code in the *@script* parameter of [sp_execute_external_script](#).
- Your original solution includes multiple steps that might be more efficient in a production environment if executed independently, such as data preparation or feature engineering vs. model training, scoring, or reporting.
- You want to optimize performance by changing libraries, using parallel execution, or offloading some processing to SQL Server.

Step 1. Plan requirements and resources

Packages

- Determine which packages are needed and ensure that they work on SQL Server.
- Install packages in advance, in the default package library used by Machine Learning Services. User libraries are not supported.

Data sources

- If you intend to embed your code in [sp_execute_external_script](#), identify primary and secondary data sources.
 - **Primary** data sources are large datasets, such as model training data, or input data for predictions. Plan to map your largest dataset to the input parameter of [sp_execute_external_script](#).
 - **Secondary** data sources are typically smaller data sets, such as lists of factors, or additional grouping variables.

Currently, `sp_execute_external_script` supports only a single dataset as input to the stored procedure. However, you can add multiple scalar or binary inputs.

Stored procedure calls preceded by EXECUTE cannot be used as an input to [sp_execute_external_script](#). You can use queries, views, or any other valid SELECT statement.

- Determine the outputs you need. If you run code using `sp_execute_external_script`, the stored procedure can output only one data frame as a result. However, you can also output multiple scalar outputs, including plots and models in binary format, as well as other scalar values derived from code or SQL parameters.

Data types

For a detailed look at the data type mappings between R/Python and SQL Server, see these articles:

- [Data type mappings between R and SQL Server](#)
- [Data type mappings between Python and SQL Server](#)

Take a look at the data types used in your R/Python code and do the following:

- Make a checklist of possible data type issues.

All R/Python data types are supported by SQL Server Machine Learning Services. However, SQL Server supports a greater variety of data types than does R or Python. Therefore, some implicit data type conversions are performed when moving SQL Server data to and from your code. You might need to explicitly cast or convert some data.

NULL values are supported. However, R uses the `na` data construct to represent a missing value, which is similar to a null.

- Consider eliminating dependency on data that cannot be used by R: for example, rowid and GUID data types from SQL Server cannot be consumed by R and will generate errors.

Step 2. Convert or repackage code

How much you change your code depends on whether you intend to submit the code from a remote client to run in the SQL Server compute context, or intend to deploy the code as part of a stored procedure. The latter can provide better performance and data security, though it imposes some additional requirements.

- Define your primary input data as a SQL query wherever possible to avoid data movement.
- When running code in a stored procedure, you can pass through multiple **scalar** inputs. For any parameters that you want to use in the output, add the **OUTPUT** keyword.

For example, the following scalar input `@model_name` contains the model name, which is also later modified by the R script, and output in its own column in the results:

SQL

```
-- declare a local scalar variable which will be passed into the R
script
DECLARE @local_model_name AS NVARCHAR (50) = 'DefaultModel';

-- The below defines an OUTPUT variable in the scope of the R script,
called model_name
-- Syntactically, it is defined by using the @model_name name. Be aware
that the sequence
-- of these parameters is very important. Mandatory parameters to
sp_execute_external_script
-- must appear first, followed by the additional parameter definitions
like @params, etc.
EXECUTE sp_execute_external_script @language = N'R', @script = N'
  model_name <- "Model name from R script"
  OutputDataSet <- data.frame(InputDataSet$c1, model_name)'
  , @input_data_1 = N'SELECT 1 AS c1'
  , @params = N'@model_name nvarchar(50) OUTPUT'
  , @model_name = @local_model_name OUTPUT;

-- optionally, examine the new value for the local variable:
SELECT @local_model_name;
```

- Any variables that you pass in as parameters of the stored procedure `sp_execute_external_script` must be mapped to variables in the code. By default, variables are mapped by name. All columns in the input dataset must also be mapped to variables in the script.

For example, assume your R script contains a formula like this one:

```
R  
  
formula <- ArrDelay ~ CRSDepTime + DayOfWeek + CRSDepHour:DayOfWeek
```

An error is raised if the input dataset does not contain columns with the matching names `ArrDelay`, `CRSDepTime`, `DayOfWeek`, `CRSDepHour`, and `DayOfWeek`.

- In some cases, an output schema must be defined in advance for the results.

For example, to insert the data into a table, you must use the **WITH RESULT SET** clause to specify the schema.

The output schema is also required if the script uses the argument `@parallel=1`. The reason is that multiple processes might be created by SQL Server to run the query in parallel, with the results collected at the end. Therefore, the output schema must be prepared before the parallel processes can be created.

In other cases, you can omit the result schema by using the option **WITH RESULT SETS UNDEFINED**. This statement returns the dataset from the script without naming the columns or specifying the SQL data types.

- Consider generating timing or tracking data using T-SQL rather than R/Python.

For example, you could pass the system time or other information used for auditing and storage by adding a T-SQL call that's passed through to the results, rather than generating similar data in the script.

Improve performance and security

- Avoid writing predictions or intermediate results to a file. Write predictions to a table instead to avoid data movement.
- Run all queries in advance, and review the SQL Server query plans to identify tasks that can be performed in parallel.

If the input query can be parallelized, set `@parallel=1` as part of your arguments to `sp_execute_external_script`.

Parallel processing with this flag is typically possible any time that SQL Server can work with partitioned tables or distribute a query among multiple processes and aggregate the results at the end. Parallel processing with this flag is typically not possible if you're training models using algorithms that require all data to be read, or if you need to create aggregates.

- Review your code to determine if there are steps that can be performed independently, or performed more efficiently, by using a separate stored procedure call. For example, you might get better performance by doing feature engineering or feature extraction separately and saving the values to a table.
- Look for ways to use T-SQL rather than R/Python code for set-based computations.

For example, this R solution shows how user-defined T-SQL functions and R can perform the same feature engineering task: [Data Science End-to-End Walkthrough](#).

- Consult with a database developer to determine ways to improve performance by using SQL Server features such as [memory-optimized tables](#), or, if you have Enterprise Edition, [Resource Governor](#).
- If you're using R, then if possible replace conventional R functions with **RevoScaleR** functions that support distributed execution. For more information, see [Comparison of Base R and RevoScaleR Functions](#).

Step 3. Prepare for deployment

- Notify the administrator so that packages can be installed and tested in advance of deploying your code.

In a development environment, it might be okay to install packages as part of your code, but this is a bad practice in a production environment.

User libraries are not supported, regardless of whether you're using a stored procedure or running R/Python code in the SQL Server compute context.

Package your R/Python code in a stored procedure

- Create a T-SQL user-defined function, embedding your code using the [sp-external-script](#) statement.
- If you have complex R code, use the R package **sqlrutils** to convert your code. This package is designed to help experienced R users write good stored procedure

code. You rewrite your R code as a single function with clearly defined inputs and outputs, then use the `sqlrutils` package to generate the input and outputs in the correct format. The `sqlrutils` package generates the complete stored procedure code for you, and can also register the stored procedure in the database.

For more information and examples, see [sqlrutils \(SQL\)](#).

Integrate with other workflows

- Leverage T-SQL tools and ETL processes. Perform feature engineering, feature extraction, and data cleansing in advance as part of data workflows.

When you're working in a dedicated development environment, you might pull data to your computer, analyze the data iteratively, and then write out or display the results. However, when standalone code is migrated to SQL Server, much of this process can be simplified or delegated to other SQL Server tools.

- Use secure, asynchronous visualization strategies.

Users of SQL Server often cannot access files on the server, and SQL client tools typically do not support the R/Python graphics devices. If you generate plots or other graphics as part of the solution, consider exporting the plots as binary data and saving to a table, or writing.

- Wrap prediction and scoring functions in stored procedures for direct access by applications.

Next steps

To view examples of how R and Python solutions can be deployed in SQL Server, see these tutorials:

R tutorials

- [Develop a predictive model in R with SQL machine learning](#)
- [Predict NYC taxi fares with binary classification](#)
- [SQL development for R data scientists](#)





Python tutorials

- [Predict ski rental with linear regression with SQL machine learning](#)

- Predict NYC taxi fares with binary classification

Native scoring using the PREDICT T-SQL function with SQL machine learning

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Database](#)  [Azure SQL Managed Instance](#)  [Azure Synapse Analytics](#)

Learn how to use native scoring with the [PREDICT T-SQL function](#) to generate prediction values for new data inputs in near-real-time. Native scoring requires that you have an already-trained model.

The `PREDICT` function uses the native C++ extension capabilities in [SQL machine learning](#). This methodology offers the fastest possible processing speed of forecasting and prediction workloads and support models in [Open Neural Network Exchange \(ONNX\)](#) [↗](#) format or models trained using the [RevoScaleR](#) and [revoscalepy](#) packages.

How native scoring works

Native scoring uses libraries that can read models in ONNX or a predefined binary format, and generate scores for new data inputs that you provide. Because the model is trained, deployed, and stored, it can be used for scoring without having to call the R or Python interpreter. This means that the overhead of multiple process interactions is reduced, resulting in faster prediction performance.

To use native scoring, call the `PREDICT` T-SQL function and pass the following required inputs:

- A compatible model based on a supported model and algorithm.
- Input data, typically defined as a T-SQL query.

The function returns predictions for the input data, together with any columns of source data that you want to pass through.

Prerequisites

`PREDICT` is available on:

- All editions of SQL Server 2017 and later on Windows and Linux
- Azure SQL Managed Instance
- Azure SQL Database

- Azure SQL Edge
- Azure Synapse Analytics

The function is enabled by default. You do not need to install R or Python, or enable additional features.

Supported models

The model formats supported by the `PREDICT` function depends on the SQL platform on which you perform native scoring. See the table below to see which model formats are supported on which platform.

Platform	ONNX model format	RevoScale model format
SQL Server	No	Yes
Azure SQL Managed Instance	Yes	Yes
Azure SQL Database	No	Yes
Azure SQL Edge	Yes	No
Azure Synapse Analytics	Yes	No

RevoScale models

The model must be trained in advance using one of the supported rx algorithms listed below using the [RevoScaleR](#) or [revoscalepy](#) package.

Serialize the model using [rxSerialize](#) for R, and [rx_serialize_model](#) for Python. These serialization functions have been optimized to support fast scoring.

Supported RevoScale algorithms

The following algorithms are supported in [revoscalepy](#) and [RevoScaleR](#).

- [revoscalepy](#) algorithms
 - [rx_lin_mod](#)
 - [rx_logit](#)
 - [rx_btrees](#)
 - [rx_dtree](#)
 - [rx_dforest](#)
- [RevoScaleR](#) algorithms

- [rxLinMod](#)
- [rxLogit](#)
- [rxBTrees](#)
- [rxDtree](#)
- [rxDForest](#)

If you need to use an algorithms from MicrosoftML or microsoftml, use [real-time scoring with sp_rxPredict](#).

Unsupported model types include the following types:

- Models containing other transformations
- Models using the `rxGlm` or `rxNaiveBayes` algorithms in RevoScaleR or revoscalepy equivalents
- PMML models
- Models created using other open-source or third-party libraries

Examples

PREDICT with RevoScale model

In this example, you create a model using **RevoScaleR** in R, and then call the real-time prediction function from T-SQL.

Step 1. Prepare and save the model

Run the following code to create the sample database and required tables.

SQL

```
CREATE DATABASE NativeScoringTest;
GO
USE NativeScoringTest;
GO
DROP TABLE IF EXISTS iris_rx_data;
GO
CREATE TABLE iris_rx_data (
    "Sepal.Length" float not null, "Sepal.Width" float not null
, "Petal.Length" float not null, "Petal.Width" float not null
, "Species" varchar(100) null
);
GO
```

Use the following statement to populate the data table with data from the **iris** dataset.

SQL

```
INSERT INTO iris_rx_data ("Sepal.Length", "Sepal.Width", "Petal.Length",
"Petal.Width" , "Species")
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N'iris_data <- iris;'
    , @input_data_1 = N''
    , @output_data_1_name = N'iris_data';
GO
```

Now, create a table for storing models.

SQL

```
DROP TABLE IF EXISTS ml_models;
GO
CREATE TABLE ml_models ( model_name nvarchar(100) not null primary key
    , model_version nvarchar(100) not null
    , native_model_object varbinary(max) not null);
GO
```

The following code creates a model based on the iris dataset and saves it to the table named **models**.

SQL

```
DECLARE @model varbinary(max);
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N'
        iris.sub <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
        iris.dtree <- rxDTree(Species ~ Sepal.Length + Sepal.Width +
Petal.Length + Petal.Width, data = iris[iris.sub, ])
        model <- rxSerializeModel(iris.dtree, realtimeScoringOnly = TRUE)
    '
    , @params = N'@model varbinary(max) OUTPUT'
    , @model = @model OUTPUT
INSERT [dbo].[ml_models]([model_name], [model_version],
[native_model_object])
VALUES('iris.dtree', 'v1', @model) ;
```

ⓘ Note

Be sure to use the `rxSerializeModel` function from `RevoScaleR` to save the model. The standard R `serialize` function cannot generate the required format.

You can run a statement such as the following to view the stored model in binary format:

SQL

```
SELECT *, datalength(native_model_object)/1024. as model_size_kb
FROM ml_models;
```

Step 2. Run PREDICT on the model

The following simple PREDICT statement gets a classification from the decision tree model using the **native scoring** function. It predicts the iris species based on attributes you provide, petal length and width.

SQL

```
DECLARE @model varbinary(max) = (
    SELECT native_model_object
    FROM ml_models
    WHERE model_name = 'iris.dtree'
    AND model_version = 'v1');
SELECT d.*, p.*
FROM PREDICT(MODEL = @model, DATA = dbo.iris_rx_data as d)
WITH(setosa_Pred float, versicolor_Pred float, virginica_Pred float) as p;
go
```

If you get the error, "Error occurred during execution of the function PREDICT. Model is corrupt or invalid", it usually means that your query didn't return a model. Check whether you typed the model name correctly, or if the models table is empty.

ⓘ Note

Because the columns and values returned by **PREDICT** can vary by model type, you must define the schema of the returned data by using a **WITH** clause.

Next steps

- [PREDICT T-SQL function](#)
- [SQL machine learning documentation](#)
- [Machine learning and AI with ONNX in SQL Edge](#)
- [Deploy and make predictions with an ONNX model in Azure SQL Edge](#)
- [Score machine learning models with PREDICT in Azure Synapse Analytics](#)

Real-time scoring with `sp_rxPredict` in SQL Server

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Learn how to perform real-time scoring with the `sp_rxPredict` system stored procedure in SQL Server for high-performance predictions or scores in forecasting workloads.

Real-time scoring with `sp_rxPredict` is language-agnostic and executes with no dependencies on the R or Python runtimes in [Machine Learning Services](#). Using a model created and trained using Microsoft functions and serialized to a binary format in SQL Server, you can use real-time scoring to generate predicted outcomes on new data inputs on SQL Server instances that do not have the R or Python add-on installed.

How real-time scoring works

Real-time scoring is supported on specific model types based on functions in [RevoScaleR](#) or [MicrosoftML](#) in R, or [revoscalepy](#) or [microsoftml](#) in Python. It uses native C++ libraries to generate scores based on user input provided to a machine learning model stored in a special binary format.

Because a trained model can be used for scoring without having to call an external language runtime in [Machine Learning Services](#), the overhead of multiple processes is reduced.

Real-time scoring is a multi-step process:

1. You enable the stored procedure that does scoring on a per-database basis.
2. You load the pre-trained model in binary format.
3. You provide new input data to be scored, either tabular or single rows, as input to the model.
4. To generate scores, call the `sp_rxPredict` stored procedure.

Prerequisites

- [Enable SQL Server CLR integration](#).
- [Enable real-time scoring](#).

- The model must be trained in advance using one of the supported rx algorithms. For details, see [Supported algorithms](#) for `sp_rxPredict`.
- Serialize the model using `rxSerialize` for R or `rx_serialize_model` for Python. These serialization functions have been optimized to support fast scoring.
- Save the model to the database engine instance from which you want to call it. This instance is not required to have the R or Python runtime extension.

ⓘ Note

Real-time scoring is currently optimized for fast predictions on smaller data sets, ranging from a few rows to hundreds of thousands of rows. On big datasets, using `rxPredict` might be faster.

Enable real-time scoring

Enable this feature for each database that you want to use for scoring. The server administrator should run the command-line utility, `RegisterRExt.exe`, which is included with the `RevoScaleR` package.

⊗ Caution

In order for real-time scoring to work, SQL CLR functionality needs to be enabled in the instance and the database needs to be marked trustworthy. When you run the script, these actions are performed for you. However, consider carefully the additional security implications before doing this.

1. Open an elevated command prompt, and navigate to the folder where `RegisterRExt.exe` is located. The following path can be used in a default installation:

```
<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\
```

2. Run the following command, substituting the name of your instance and the target database where you want to enable the extended stored procedures:

```
RegisterRExt.exe /installRts [/instance:name] /database:databasename
```

For example, to add the extended stored procedure to the `CLRpredict` database on the default instance, type:

```
RegisterRExt.exe /installRts /database:CLRpredict
```

The instance name is optional if the database is on the default instance. If you're using a named instance, specify the instance name.

3. RegisterRExt.exe creates the following objects:

- Trusted assemblies
- The stored procedure `sp_rxPredict`
- A new database role, `rxpredict_users`. The database administrator can use this role to grant permission to users who use the real-time scoring functionality.

4. Add any users who need to run `sp_rxPredict` to the new role.

ⓘ Note

In SQL Server 2017 and later, additional security measures are in place to prevent problems with CLR integration. These measures impose additional restrictions on the use of this stored procedure as well.

Disable real-time scoring

To disable real-time scoring functionality, open an elevated command prompt, and run the following command: `RegisterRExt.exe /uninstallrts /database:<database_name> [/instance:name]`

Example

This example describes the steps required to prepare and save a model for **real-time** prediction, and provides an example in R of how to call the function from T-SQL.

Step 1. Prepare and save the model

The binary format required by `sp_rxPredict` is the same as the format required to use the `PREDICT` function. Therefore, in your R code, include a call to `rxSerializeModel`, and be sure to specify `realtimeScoringOnly = TRUE`, as in this example:

```
R
```

```
model <- rxSerializeModel(model.name, realtimeScoringOnly = TRUE)
```

Step 2. Call sp_rxPredict

You call `sp_rxPredict` as you would any other stored procedure. In the current release, the stored procedure takes only two parameters: `@model` for the model in binary format, and `@inputData` for the data to use in scoring, defined as a valid SQL query.

Because the binary format is the same as that used by the PREDICT function, you can use the models and data table from the preceding example.

SQL

```
DECLARE @irismodel varbinary(max)
SELECT @irismodel = [native_model_object] from [ml_models]
WHERE model_name = 'iris.dtree'
AND model_version = 'v1'

EXEC sp_rxPredict
@model = @irismodel,
@inputData = N'SELECT * FROM iris_rx_data'
```

ⓘ Note

The call to `sp_rxPredict` fails if the input data for scoring does not include columns that match the requirements of the model. Currently, only the following .NET data types are supported: double, float, short, ushort, long, ulong and string.

Therefore, you might need to filter out unsupported types in your input data before using it for real-time scoring.

For information about corresponding SQL types, see [SQL-CLR Type Mapping](#) or [Mapping CLR Parameter Data](#).

Next steps

- [Native scoring using the PREDICT T-SQL function with SQL machine learning](#)
- [sp_rxPredict](#)
- [SQL machine learning](#)

Get Python package information

Article • 02/28/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

This article describes how to get information about installed Python packages, including versions and installation locations, on [SQL Server Machine Learning Services](#). Example Python scripts show you how to list package information such as installation path and version.

Default Python library location

When you install machine learning with SQL Server, a single package library is created at the instance level for each language that you install. The instance library is a secured folder registered with SQL Server.

All script or code that runs in-database on SQL Server must load functions from the instance library. SQL Server can't access packages installed to other libraries. This applies to remote clients as well: any Python code running in the server compute context can only use packages installed in the instance library. To protect server assets, the default instance library can be modified only by a computer administrator.

The default path of the binaries for Python is:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES
```

This assumes the default SQL instance, MSSQLSERVER. If SQL Server is installed as a user-defined named instance, the given name is used instead.

Enable external scripts by running the following SQL commands:

SQL

```
sp_configure 'external scripts enabled', 1;  
RECONFIGURE WITH override;
```

Run the following SQL statement if you want to verify the default library for the current instance. This example returns the list of folders included in the Python `sys.path` variable. The list includes the current directory and the standard library path.

SQL

```
EXECUTE sp_execute_external_script
@language =N'Python',
@script=N'import sys; print("\n".join(sys.path))'
```

For more information about the variable `sys.path` and how it's used to set the interpreter's search path for modules, see [The Module Search Path](#).

Default Microsoft Python packages

The following Microsoft Python packages are installed with SQL Server Machine Learning Services when you select the Python feature during setup.

Packages	Version	Description
revoscalepy	9.4.7	Used for remote compute contexts, streaming, parallel execution of rx functions for data import and transformation, modeling, visualization, and analysis.
microsoftml	9.4.7	Adds machine learning algorithms in Python.

For information on which version of Python is included, see [Python and R versions](#).

Component upgrades

By default, Python packages are refreshed through service packs and cumulative updates. Additional packages and full version upgrades of core Python components are possible only through product upgrades.

Default open-source Python packages

When you select the Python language option during setup, Anaconda 4.2 distribution (over Python 3.5) is installed. In addition to Python code libraries, the standard installation includes sample data, unit tests, and sample scripts.

Important

You should never manually overwrite the version of Python installed by SQL Server Setup with newer versions on the web. Microsoft Python packages are based on specific versions of Anaconda. Modifying your installation could destabilize it.

List all installed Python packages

The following example script displays a list of all Python packages installed in the SQL Server instance.

SQL

```
EXECUTE sp_execute_external_script
    @language = N'Python',
    @script = N'
import pkg_resources
import pandas
OutputDataSet = pandas.DataFrame(sorted([(i.key, i.version) for i in
pkg_resources.working_set]))'
WITH result sets((Package NVARCHAR(128), Version NVARCHAR(128)));
```

Find a single Python package

If you've installed a Python package and want to make sure that it's available to a particular SQL Server instance, you can execute a stored procedure to look for the package and return messages.

For example, the following code looks for the `scikit-learn` package. If the package is found, the code prints the package version.

SQL

```
EXECUTE sp_execute_external_script
    @language = N'Python',
    @script = N'
import pkg_resources
pkg_name = "scikit-learn"
try:
    version = pkg_resources.get_distribution(pkg_name).version
    print("Package " + pkg_name + " is version " + version)
except:
    print("Package " + pkg_name + " not found")
'
```

Result:

text

```
STDOUT message(s) from external script: Package scikit-learn is version
0.20.2
```

View the version of Python

The following example code returns the version of Python installed in the instance of SQL Server.

SQL


```
EXECUTE sp_execute_external_script
    @language = N'Python',
    @script = N'
import sys
print(sys.version)
'
```

Next steps

- [Install packages with Python tools](#)

Install packages with Python tools on SQL Server

Article • 02/28/2023

Applies to:  SQL Server 2017 (14.x) only

This article describes how to use standard Python tools to install new Python packages on an instance of SQL Server Machine Learning Services. In general, the process for installing new packages is similar to that in a standard Python environment. However, some additional steps are required if the server does not have an Internet connection.

For more information about package location and installation paths, see [Get Python package information](#).

Prerequisites

- You must have [SQL Server Machine Learning Services](#) installed with the Python language option.

Other considerations

- Packages must be Python 3.5-compliant and run on Windows.
- The Python package library is located in the Program Files folder of your SQL Server instance and, by default, installing in this folder requires administrator permissions. For more information, see [Package library location](#).
- Package installation is per instance. If you have multiple instances of Machine Learning Services, you must add the package to each one.
- Database servers are frequently locked down. In many cases, Internet access is blocked entirely. For packages with a long list of dependencies, you will need to identify these dependencies in advance and be ready to install each one manually.
- Before adding a package, consider whether the package is a good fit for the SQL Server environment.
 - We recommend that you use Python in-database for tasks that benefit from tight integration with the database engine, such as machine learning, rather than tasks that simply query the database.

- If you add packages that put too much computational pressure on the server, performance will suffer.
- On a hardened SQL Server environment, you might want to avoid the following:
 - Packages that require network access
 - Packages that require elevated file system access
 - Packages used for web development or other tasks that don't benefit by running inside SQL Server

Add a Python package on SQL Server

To install a new Python package that can be used in a script on SQL Server, you install the package in the instance of Machine Learning Services. If you have multiple instances of Machine Learning Services, you must add the package to each one.

The package installed in the following examples is [CNTK](#), a framework for deep learning from Microsoft that supports customization, training, and sharing of different types of neural networks.

For offline install, download the Python package

If you are installing Python packages on a server with no Internet access, you must download the WHL file from a computer with Internet access and then copy the file to the server.

For example, on an Internet-connected computer you can download a `.whl` file for CNTK and then copy the file to a local folder on the SQL Server computer. See [Install CNTK from Wheel Files](#) for a list of available `.whl` files for CNTK.

Important

Make sure that you get the Windows version of the package. If the file ends in `.gz`, it's probably not the right version.

For more information about downloads of the CNTK framework for multiple platforms and for multiple versions of Python, see [Setup CNTK on your machine](#).

Locate the Python library

Locate the default Python library location used by SQL Server. If you have installed multiple instances, locate the `PYTHON_SERVICES` folder for the instance where you want to

add the package.

For example, if Machine Learning Services was installed using defaults, and machine learning was enabled on the default instance, the path is:

Console

```
cd "C:\Program Files\Microsoft SQL
Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES"
```

💡 Tip

For future debugging and testing, you might want to set up a Python environment specific to the instance library.

Install the package using pip

Use the `pip` installer to install new packages. You can find `pip.exe` in the `Scripts` subfolder of the `PYTHON_SERVICES` folder. SQL Server Setup does not add the `Scripts` subfolder to the system path, so you must specify the full path, or you can add the `Scripts` folder to the `PATH` variable in Windows.

ⓘ Note

If you're using Visual Studio 2017, or Visual Studio 2015 with the Python extensions, you can run `pip install` from the **Python Environments** window. Click **Packages**, and in the text box, provide the name or location of the package to install. You don't need to type `pip install`; it is filled in for you automatically.

- If the computer has Internet access, provide the name of the package:

Console

```
scripts\pip.exe install cntk
```

You can also specify the URL of a specific package and version, for example:

Console

```
scripts\pip.exe install https://cntk.ai/PythonWheel/CPU-Only/cntk-2.1-
cp35-cp35m-win_amd64.whl
```

- If the computer does not have Internet access, specify the WHL file you downloaded earlier. For example:

Console

```
scripts\pip.exe install C:\Downloads\cntk-2.1-cp35-cp35m-win_amd64.whl
```

You might be prompted to elevate permissions to complete the install. As the installation progresses, you can see status messages in the command prompt window.

Load the package or its functions as part of your script

When installation is complete, you can immediately begin using the package in Python scripts in SQL Server.

To use functions from the package in your script, insert the standard `import` `<package_name>` statement in the initial lines of the script:

SQL

```
EXECUTE sp_execute_external_script
  @language = N'Python',
  @script = N'
import cntk
# Python statements ...
'
```

See also

- [Get Python package information](#)
- [Python tutorials for SQL Server Machine Learning Services](#)
- [Python API for CNTK](#) ↗.

Get R package information

Article • 08/01/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

This article describes how to get information about installed R packages on [SQL Server Machine Learning Services](#). Example R scripts show you how to list package information such as installation path and version.

Note

Feature capabilities and installation options vary between versions of SQL Server. Use the version selector dropdown to choose the appropriate version of SQL Server.

Default R library location

When you install machine learning with SQL Server, a single package library is created at the instance level for each language that you install. On Windows, the instance library is a secured folder registered with SQL Server.

All script that runs in-database on SQL Server must load functions from the instance library. SQL Server can't access packages installed to other libraries. This applies to remote clients as well: any R script running in the server compute context can only use packages installed in the instance library. To protect server assets, the default instance library can be modified only by a computer administrator.

The default path of the binaries for R is:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library
```

This assumes the default SQL instance, MSSQLSERVER. If SQL Server is installed as a user-defined named instance, the given name is used instead.

Run the following statement to verify the default R package library for the current instance:

SQL

```
EXECUTE sp_execute_external_script
  @language = N'R',
  @script = N'OutputDataSet <- data.frame(.libPaths());'
```

```
WITH RESULT SETS (([DefaultLibraryName] VARCHAR(MAX) NOT NULL));  
GO
```

Default Microsoft R packages

The following Microsoft R packages are installed with SQL Server Machine Learning Services when you select the R feature during setup.

Packages	Version	Description
RevoScaleR	9.2	Used for remote compute contexts, streaming, parallel execution of rx functions for data import and transformation, modeling, visualization, and analysis.
sqlrutils	1.0.0	Used for including R script in stored procedures.
MicrosoftML	1.4.0	Adds machine learning algorithms in R.
olapR	1.0.0	Used for writing MDX statements in R.

Component upgrades

By default, R packages are refreshed through service packs and cumulative updates. Additional packages and full version upgrades of core R components are possible only through product upgrades.

Default open-source R packages

R support includes open-source R so that you can call base R functions and install additional open-source and third-party packages. R language support includes core functionality such as **base**, **stats**, **utils**, and others. A base installation of R also includes numerous sample datasets and standard R tools such as **RGui** (a lightweight interactive editor) and **RTerm** (an R command prompt).

For information on which version of R is included with each SQL Server version, see [Python and R versions](#).

Important

You should never manually overwrite the version of R installed by SQL Server Setup with newer versions on the web. Microsoft R packages are based on specific versions of R. Modifying your installation could destabilize it.

List all installed R packages

The following example uses the R function `installed.packages()` in a Transact-SQL stored procedure to display a list of R packages that have been installed in the `R_SERVICES` library for the current SQL instance. This script returns package name and version fields in the `DESCRIPTION` file.

SQL

```
EXECUTE sp_execute_external_script
    @language=N'R',
    @script = N'str(OutputDataSet);
    packagematrix <- installed.packages();
    Name <- packagematrix[,1];
    Version <- packagematrix[,3];
    OutputDataSet <- data.frame(Name, Version);',
    @input_data_1 = N'
    '
WITH RESULT SETS ((PackageName nvarchar(250), PackageVersion nvarchar(max)
))
```

For more information about the optional and default fields for the R package `DESCRIPTION` field, see <https://cran.r-project.org>.

Find a single R package

If you've installed an R package and want to make sure that it's available to a particular SQL Server instance, you can execute a stored procedure to load the package and return messages.

For example, the following statement looks for and loads the [glue](#) package, if available. If the package cannot be located or loaded, you get an error.

SQL

```
EXECUTE sp_execute_external_script
    @language =N'R',
    @script=N'
    require("glue")
    '
```

To see more information about the package, view the `packageDescription`. The following statement returns information for the `MicrosoftML` package.

SQL



```
EXECUTE sp_execute_external_script
  @language = N'R',
  @script = N'
print(packageDescription("MicrosoftML"))
'
```

Next steps

- [Install packages with R tools](#)

Install packages with R tools

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x)  SQL Server 2017 (14.x)

This article describes how to use standard R tools to install new R packages to an instance of SQL Server Machine Learning Services or SQL Server R Services. You can install packages on a SQL Server that has an Internet connection, as well as one that is isolated from the Internet.

In addition to standard R tools, you can install R packages using:

- [RevoScaleR](#)
- [T-SQL \(CREATE EXTERNAL LIBRARY\)](#)

General considerations

- R code running in SQL Server can use only packages installed in the default instance library. SQL Server cannot load packages from external libraries, even if that library is on the same computer. This includes R libraries installed with other Microsoft products.
- The R package library is located in the Program Files folder of your SQL Server instance and, by default, installing in this folder requires administrator permissions. For more information, see [Package library location](#).

Non-administrators can install packages using RevoScaleR 9.0.1 and later, or using CREATE EXTERNAL LIBRARY. The **dbo_owner** user, or a user with CREATE EXTERNAL LIBRARY permission, can install R packages to the current database. For more information, see:

- [Use RevoScaleR to install R packages](#)
- [Use T-SQL \(CREATE EXTERNAL LIBRARY\) to install R packages on SQL Server](#)
- On a hardened SQL Server environment, you might want to avoid the following:
 - Packages that require network access
 - Packages that require elevated file system access
 - Packages used for web development or other tasks that don't benefit by running inside SQL Server

Online installation (with Internet access)

If the SQL Server has access to the Internet, then you can use standard package installation tools to install R packages.

1. Determine the location of the instance library (see [Get R package information](#)) and navigate to the folder where the R tools are installed.

For example the default path for a SQL Server default instance is:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\bin\x64\
```

2. Run **R** or **Rgui** as administrator from this folder.
3. Run the R command `install.packages` and specify the package name. If the package has any dependencies, the installer automatically downloads the dependencies and installs them.

If you have multiple, side-by-side instances of SQL Server, run the installation separately for each instance in which you want to use the package. Packages cannot be shared across instances.

Offline installation (no internet access)

Frequently, servers that host production databases don't have an internet connection. To install R packages in that environment, you download and prepare packages and dependencies in advance (as zipped files), and then copy the files to a folder on the server. Once the files are in place, the packages can be installed offline.

Identifying all dependencies gets complicated. For R, we recommend that you use [miniCRAN](#) to create a local repository. **miniCRAN** takes a list of packages you want to install, analyzes dependencies, and collects all the necessary zipped files. It then creates a single repository that you can copy to the isolated SQL Server instance. The [igraph](#) package is also helpful in analyzing package dependencies.

For more information, see [Create a local R package repository using miniCRAN](#).

Once the zip file is on the SQL Server instance, you can install it using standard R tools on the server.

1. Determine the location of the instance library (see [Get R package information](#)) and navigate to the folder where the R tools are installed.

For example the default path for a SQL Server default instance is:

```
C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\bin\x64\
```

2. Run **R** or **Rgui** as administrator from this folder.
3. Run the R command `install.packages` and specify the package or repository name, and the location of the zipped files. For example:

```
R  
  
install.packages("C:\\Temp\\Downloaded packages\\mynewpackage.zip",  
repos=NULL)
```

This command extracts the R package `mynewpackage` from its local zipped file and installs the package. If the package has any dependencies, the installer checks for existing packages in the library. If you have created a repository that includes the dependencies, the installer installs the required packages as well.

ⓘ Note

If any required packages are not present in the instance library, and cannot be found in the zipped files, installation of the target package fails.

As an alternative to **miniCRAN**, you can perform these steps manually:


1. Identify all package dependencies.
2. Check whether any required packages are already installed on the server. If the package is installed, verify that the version is correct.
3. Download the package and all dependencies to a separate computer with Internet access.
4. Place the package and dependencies in a single package archive.
5. Zip the archive if it's not already in zipped format.
6. Move the files to a folder accessible by the server.
7. Run a supported installation command or DDL statement to install the package into the instance library.

See also

- [Get R package information](#)
- [Tips for using R packages](#)
- [SQL Server R language tutorials](#)

Use T-SQL (CREATE EXTERNAL LIBRARY) to install R packages on SQL Server

Article • 02/28/2023

Applies to:  SQL Server 2017 (14.x) only



This article explains how to install new R packages on an instance of SQL Server where machine learning is enabled. There are multiple approaches to choose from. Using T-SQL works best for server administrators who are unfamiliar with R.

The [CREATE EXTERNAL LIBRARY](#) statement makes it possible to add a package or set of packages to an instance or a specific database without running R or Python code directly. However, this method requires package preparation and additional database permissions.

- All packages must be available as a local zipped file, rather than downloaded on demand from the internet.
- All dependencies must be identified by name and version, and included in the zip file. The statement fails if required packages are not available, including downstream package dependencies.
- You must be **db_owner** or have [CREATE EXTERNAL LIBRARY](#) permission in a database role. For details, see [CREATE EXTERNAL LIBRARY](#).

Download packages in archive format

If you are installing a single package, download the package in zipped format.

It's more common to install multiple packages due to package dependencies. When a package requires other packages, you must verify that all of them are accessible to each other during installation. We recommend [creating a local repository](#) using [miniCRAN](#)  to assemble a full collection of packages, as well as [igraph](#)  for analyzing packages dependencies. Installing the wrong version of a package or omitting a package dependency can cause a [CREATE EXTERNAL LIBRARY](#) statement to fail.

Copy the file to a local folder

Copy the zipped file containing all packages to a local folder on the server. If you do not have access to the file system on the server, you can also pass a complete package as a

variable, using a binary format. For more information, see [CREATE EXTERNAL LIBRARY](#).

Run the statement to upload packages

Open a **Query** window, using an account with administrative privileges.

Run the T-SQL statement `CREATE EXTERNAL LIBRARY` to upload the zipped package collection to the database.

For example, the following statement names as the package source a miniCRAN repository containing the **randomForest** package, together with its dependencies.

SQL

```
CREATE EXTERNAL LIBRARY [randomForest]
FROM (CONTENT = 'C:\Temp\Rpackages\randomForest_4.6-12.zip')
WITH (LANGUAGE = 'R');
```

You cannot use an arbitrary name; the external library name must have the same name that you expect to use when loading or calling the package.

Verify package installation

If the library is successfully created, you can run the package in SQL Server, by calling it inside a stored procedure.

SQL



```
EXEC sp_execute_external_script
@language =N'R',
@script=N'library(randomForest)'
```

See also

- [Get R package information](#)
- [R tutorials](#)

Use RevoScaleR to install R packages

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x)  SQL Server 2017 (14.x)

This article describes how to use [RevoScaleR](#) (version 9.0.1 and later) functions to install R packages on SQL Server with Machine Learning Services or R Services. The RevoScaleR functions can be used by remote, non-administrators to install packages on SQL Server without direct access to the server.

RevoScaleR functions for package management

The following table describes the functions used for R package installation and management.

Function	Description
rxSqlLibPaths	Determine the path of the instance library on the remote SQL Server.
rxFindPackage	Gets the path for one or more packages on the remote SQL Server.
rxInstallPackages	Call this function from a remote R client to install packages in a SQL Server compute context, either from a specified repository, or by reading locally saved zipped packages. This function checks for dependencies and ensures that any related packages can be installed to SQL Server, just like R package installation in the local compute context. To use this option, you must have enabled package management on the server and database. Both client and server environments must have the same version of RevoScaleR.
rxInstalledPackages	Gets a list of packages installed in the specified compute context.
rxSyncPackages	Copy information about a package library between the file system and database, for the specified compute context.
rxRemovePackages	Removes packages from a specified compute context. It also computes dependencies and ensures that packages that are no longer used by other packages on SQL Server are removed, to free up resources.

Prerequisites

- Remote management enabled on SQL Server. For more information, see [Enable remote R package management on SQL Server](#).

- RevoScaleR versions are the same on both client and server environments. For more information, see [Get R package information](#).
- You have permission to connect to the server and a database, and to run R commands. You must be a member of a database role that allows you to install packages on the specified instance and database.
 - Packages in **shared scope** can be installed by users belonging to the `rpkg-shared` role in a specified database. All users in this role can uninstall shared packages.
 - Packages in **private scope** can be installed by any user belonging to the `rpkg-private` role in a database. However, users can see and uninstall only their own packages.
 - Database owners can work with shared or private packages.

Client connections

Important

The support for Machine Learning Server (previously known as R Server) ended on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

A client workstation can be [Microsoft R Client](#) or a [Microsoft Machine Learning Server](#) (data scientists often use the free developer edition) on the same network.

When calling package management functions from a remote R client, you must create a compute context object first, using the [RxInSqlServer](#) function. Thereafter, for each package management function that you use, pass the compute context as an argument.

User identity is typically specified when setting the compute context. If you don't specify a user name and password when you create the compute context, the identity of the user running the R code is used.

1. From an R command line, define a connection string to the instance and database.
2. Use the [RxInSqlServer](#) constructor to define a SQL Server compute context, using the connection string.

```
sqlcc <- RxInSqlServer(connectionString = myConnString, shareDir =  
sqlShareDir, wait = sqlWait, consoleOutput = sqlConsoleOutput)
```

3. Create a list of the packages you want to install and save the list in a string variable.

```
R
```

```
packageList <- c("e1071", "mice")
```

4. Call `rxInstallPackages` and pass the compute context and the string variable containing the package names.

```
R
```

```
rxInstallPackages(pkgs = packageList, verbose = TRUE, computeContext =  
sqlcc)
```

If dependent packages are required, they are also installed, assuming an internet connection is available on the client.

Packages are installed using the credentials of the user making the connection, in the default scope for that user.

Call package management functions in stored procedures

You can run package management functions inside `sp_execute_external_script`. When you do so, the function is executed using the security context of the stored procedure caller.

Examples

This section provides examples of how to use these functions from a remote client when connecting to a SQL Server instance or database as the compute context.

For all examples, you must provide either a connection string, or a compute context, which requires a connection string. This example provides one way to create a compute context for SQL Server:

```
R
```

```
instance_name <- "computer-name/instance-name";
database_name <- "TestDB";
sqlWait= TRUE;
sqlConsoleOutput <- TRUE;
connString <- paste("Driver=SQL Server;Server=", instance_name,
";Database=", database_name, ";Trusted_Connection=true;", sep="");
sqlcc <- RxInSqlServer(connectionString = connString, wait = sqlWait,
consoleOutput = sqlConsoleOutput, numTasks = 4);
```

Depending on where the server is located, and the security model, you might need to provide a domain and subnet specification in the connection string, or use a SQL login. For example:

R

```
connStr <- "Driver=SQL
Server;Server=myserver.financeweb.contoso.com;Database=Finance;Uid=RUser1;Pw
d=RUserPassword"
```

Get package path on a remote SQL Server compute context

This example gets the path for the **RevoScaleR** package on the compute context, `sqlcc`.

R

```
sqlPackagePaths <- rxFindPackage(package = "RevoScaleR", computeContext =
sqlcc)
print(sqlPackagePaths)
```

Results

```
"C:/Program Files/Microsoft SQL
Server/MSSQL14.MSSQLSERVER/R_SERVICES/library/RevoScaleR"
```

Tip

If you have enabled the option to see SQL console output, you might get status messages from the function that precedes the `print` statement. After you have finished testing your code, set `consoleOutput` to `FALSE` in the compute context constructor to eliminate messages.

Get locations for multiple packages

The following example gets the paths for the `RevoScaleR` and `lattice` packages, on the compute context, `sqlcc`. To get information about multiple packages, pass a string vector containing the package names.

R

```
packagePaths <- rxFindPackage(package = c("RevoScaleR", "lattice"),  
computeContext = sqlcc)  
print(packagePaths)
```

Get package versions on a remote compute context

Run this command from an R console to get the build number and version numbers for packages installed on the compute context, `sqlServer`.

R

```
sqlPackages <- rxInstalledPackages(fields = c("Package", "Version",  
"Built"), computeContext = sqlServer)
```

Install a package on SQL Server

This example installs the `forecast` package and its dependencies into the compute context.

R

```
pkgs <- c("forecast")  
rxInstallPackages(pkgs = pkgs, verbose = TRUE, scope = "private",  
computeContext = sqlcc)
```

Remove a package from SQL Server

This example removes the `forecast` package and its dependencies from the compute context.

R

```
pkgs <- c("forecast")  
rxRemovePackages(pkgs = pkgs, verbose = TRUE, scope = "private",  
computeContext = sqlcc)
```

Synchronize packages between database and file system

The following example checks the database `TestDB`, and determines whether all packages are installed in the file system. If some packages are missing, they are installed in the file system.

R

```
# Instantiate the compute context
connectionString <- "Driver=SQL
Server;Server=myServer;Database=TestDB;Trusted_Connection=True;"
computeContext <- RxInSqlServer(connectionString = connectionString )

# Synchronize the packages in the file system for all scopes and users
rxSyncPackages(computeContext=computeContext, verbose=TRUE)
```

Package synchronization works on a per database and per user basis. For more information, see [R package synchronization for SQL Server](#).

Use a stored procedure to list packages in SQL Server

Run this command from Management Studio or another tool that supports T-SQL, to get a list of installed packages on the current instance, using `rxInstalledPackages` in a stored procedure.

SQL

```
EXEC sp_execute_external_script
  @language=N'R',
  @script=N'
    myPackages <- rxInstalledPackages();
    OutputDataSet <- as.data.frame(myPackages);
  '
```

The `rxSqlLibPaths` function can be used to determine the active library used by SQL Server Machine Learning Services. This script can return only the library path for the current server.

SQL

```
declare @instance_name nvarchar(100) = @@SERVERNAME, @database_name
nvarchar(128) = db_name();
exec sp_execute_external_script
  @language = N'R',
```



```
@script = N'  
  connStr <- paste("Driver=SQL Server;Server=", instance_name,  
";Database=", database_name, ";Trusted_Connection=true;", sep="");  
  .libPaths(rxSqlLibPaths(connStr));  
  print(.libPaths());  
' ,  
@input_data_1 = N' ',  
@params = N'@instance_name nvarchar(100), @database_name nvarchar(128)',  
@instance_name = @instance_name,  
@database_name = @database_name;
```

See also

- [Enable remote R package management](#)
- [Synchronize R packages](#)
- [Tips for using R packages](#)
- [Get R package information](#)

Enable or disable remote package management for SQL Server

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x)  SQL Server 2017 (14.x)

Important

The support for Machine Learning Server (previously known as R Server) ended on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This article describes how to enable remote management of R packages from a client workstation or a different Machine Learning Server. After the package management feature has been enabled on SQL Server, you can use RevoScaleR commands on a client to install packages on SQL Server.

By default, the external package management feature for SQL Server is disabled. You must run a separate script to enable the feature as described in the next section.

Overview of process and tools

To enable or disable package management on SQL Server, use the command-line utility **RegisterRExt.exe**, which is included with the **RevoScaleR** package.

Enabling this feature is a two-step process, requiring a database administrator: you enable package management on the SQL Server instance (once per SQL Server instance), and then enable package management on the SQL database (once per SQL Server database).

Disabling the package management feature also requires multiple steps: you remove database-level packages and permissions (once per database), and then remove the roles from the server (once per instance).

Enable package management

1. On SQL Server, open an elevated command prompt and navigate to the folder containing the utility, RegisterRExt.exe. The default location is

```
<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRExt.exe.
```

2. Run the following command, providing appropriate arguments for your environment:

```
RegisterRExt.exe /install pkgmgmt [/instance:name] [/user:username]
[/password:*|password]
```

This command creates instance-level objects on the SQL Server computer that are required for package management. It also restarts the Launchpad for the instance.

If you do not specify an instance, the default instance is used. If you do not specify a user, the current security context is used. For example, the following command enables package management on the default instance, using the credentials of the user who opened the command prompt:

```
RegisterRExt.exe /install pkgmgmt
```

3. To add package management to a specific database, run the following command from an elevated command prompt:

```
RegisterRExt.exe /install pkgmgmt /database:databasename [/instance:name]
[/user:username] [/password:*|password]
```

This command creates some database artifacts, including the following database roles that are used for controlling user permissions: `rpkgs-users`, `rpkgs-private`, and `rpkgs-shared`.

For example, the following command enables package management on the database, on the default instance. If you do not specify a user, the current security context is used.

```
RegisterRExt.exe /install pkgmgmt /database:TestDB
```

4. Repeat the command for each database where packages must be installed.
5. To verify that the new roles have been successfully created, in SQL Server Management Studio, click the database, expand **Security**, and expand **Database Roles**.

You can also run a query on `sys.database_principals` such as the following:

SQL

```
SELECT pr.principal_id, pr.name, pr.type_desc,
       pr.authentication_type_desc, pe.state_desc,
       pe.permission_name, s.name + '.' + o.name AS ObjectName
FROM sys.database_principals AS pr
```

```
JOIN sys.database_permissions AS pe
      ON pe.grantee_principal_id = pr.principal_id
JOIN sys.objects AS o
      ON pe.major_id = o.object_id
JOIN sys.schemas AS s
      ON o.schema_id = s.schema_id;
```

After you have enabled this feature, you can use RevoScaleR function to install or uninstall packages from a remote R client.

Disable package management

1. From an elevated command prompt, run the RegisterRExt utility again, and disable package management at the database level:

```
RegisterRExt.exe /uninstall pkgmgmt /database:databasename [/instance:name]
[/user:username] [/password:*|password]
```

This command removes database objects related to package management from the specified database. It also removes all the packages that were installed from the secured file system location on the SQL Server computer.

2. Repeat this command on each database where package management was used.
3. (Optional) After all databases have been cleared of packages using the preceding step, run the following command from an elevated command prompt:

```
RegisterRExt.exe /uninstall pkgmgmt [/instance:name] [/user:username]
[/password:*|password]
```


This command removes the package management feature from the instance. You might need to manually restart the Launchpad service once more to see changes.

Next steps

- [Use RevoScaleR to install R packages](#)
- [Get R package information](#)
- [Tips for using R packages](#)

R package synchronization for SQL Server

Article • 02/28/2023

Applies to:  SQL Server 2017 (14.x) only

The version of RevoScaleR included in SQL Server 2017 includes the ability to synchronize collections of R packages between the file system and the instance and database where packages are used.

This feature was provided to make it easier to back up R package collections associated with SQL Server databases. Using this feature, an administrator can restore not just the database, but any R packages that were used by data scientists working in that database.

This article describes the package synchronization feature, and how to use the `rxSyncPackages` function to perform the following tasks:

- Synchronize a list of packages for an entire SQL Server database
- Synchronize packages used by an individual user, or by a group of users
- If a user moves to a different SQL Server, you can take a backup of the user's working database and restore it to the new server, and the packages for the user will be installed into the file system on the new server, as required by R.

For example, you might use package synchronization in these scenarios:

- The DBA has restored an instance of SQL Server to a new machine and asks users to connect from their R clients and run `rxSyncPackages` to refresh and restore their packages.
- You think an R package on the file system is corrupted so you run `rxSyncPackages` on the SQL Server.

Requirements

Before you can use package synchronization, you must have the appropriate version of Microsoft R. This feature is provided in Microsoft R version 9.1.0 or later.

You must also enable the [package management feature](#) on the server.

Determine whether your server supports package management

This feature is available in SQL Server 2017 CTP 2 or later.

Enable the package management feature

To use package synchronization requires that the new package management feature be enabled on the SQL Server instance, and on individual databases. For more information, see [Enable or disable package management for SQL Server](#).

1. The server administrator enables the feature for the SQL Server instance.
2. For each database, the administrator grants individual users the ability to install or share R packages, using database roles.

When this is done, you can use RevoScaleR functions, such as [rxInstallPackages](#) to install packages into a database. Information about users and the packages that they can use is stored in the SQL Server instance.

Whenever you add a new package using the package management functions, both the records in SQL Server and the file system are updated. This information can be used to restore package information for the entire database.

Permissions

- The person who executes the package synchronization function must be a security principal on the SQL Server instance and database that has the packages.
- The caller of the function must be a member of one of these package management roles: **rpkgs-shared** or **rpkgs-private**.
- To synchronize packages marked as **shared**, the person who is running the function must have membership in the **rpkgs-shared** role, and the packages that are being moved must have been installed to a shared scope library.
- To synchronize packages marked as **private**, either the owner of the package or the administrator must run the function, and the packages must be private.
- To synchronize packages on behalf of other users, the owner must be a member of the **db_owner** database role.

How package synchronization works

To use package synchronization, call `rxSyncPackages`, which is a new function in [RevoScaleR](#).

For each call to `rxSyncPackages`, you must specify a SQL Server instance and database. Then, either list the packages to synchronize, or specify package scope.

1. Create the SQL Server compute context by using the `RxInSqlServer` function. If you don't specify a compute context, the current compute context is used.
2. Provide the name of a database on the instance in the specified compute context. Packages are synchronized per database.
3. Specify the packages to synchronize by using the scope argument.

If you use **private** scope, only packages owned by the specified owner are synchronized. If you specify **shared** scope, all non-private packages in the database are synchronized.

If you run the function without specifying either **private** or **shared** scope, all packages are synchronized.

4. If the command is successful, existing packages in the file system are added to the database, with the specified scope and owner.

If the file system is corrupted, the packages are restored based on the list maintained in the database.

If the package management feature is not available on the target database, an error is raised: "The package management feature is either not enabled on the SQL Server or version is too old"

Example 1. Synchronize all package by database

This example gets any new packages from the local file system and installs the packages in the database [TestDB]. Because no owner is specific, the list includes all packages that have been installed for private and shared scopes.

R

```
connectionString <- "Driver=SQL
Server;Server=myServer;Database=TestDB;Trusted_Connection=True;"
computeContext <- RxInSqlServer(connectionString = connectionString )
rxSyncPackages(computeContext=computeContext, verbose=TRUE)
```

Example 2. Restrict synchronized packages by scope

The following examples synchronize only the packages in the specified scope.

R

```
#Shared scope
rxSyncPackages(computeContext=computeContext, scope="shared", verbose=TRUE)

#Private scope
rxSyncPackages(computeContext=computeContext, scope="private", verbose=TRUE)
```

Example 3. Restrict synchronized packages by owner

The following example demonstrates how to synchronize only the packages that were installed for a specific user. In this example, the user is identified by the SQL login name, *user1*.

R

```
rxSyncPackages(computeContext=computeContext, scope="private", owner =
"user1", verbose=TRUE))
```

Related resources

[R package management for SQL Server](#)

Create a local R package repository using miniCRAN

Article • 08/01/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

This article describes how to install R packages offline by using [miniCRAN](#) to create a local repository of packages and dependencies. **miniCRAN** identifies and downloads packages and dependencies into a single folder that you copy to other computers for offline R package installation.

You can specify one or more packages, and **miniCRAN** recursively reads the dependency tree for these packages. It then downloads only the listed packages and their dependencies from CRAN or similar repositories.

When it's done, **miniCRAN** creates an internally consistent repository consisting of the selected packages and all required dependencies. You can move this local repository to the server, and proceed to install the packages without an internet connection.

Experienced R users often look for the list of dependent packages in the DESCRIPTION file of a downloaded package. However, packages listed in **Imports** might have second-level dependencies. For this reason, we recommend **miniCRAN** for assembling the full collection of required packages.

Why create a local repository

The goal of creating a local package repository is to provide a single location that a server administrator or other users in the organization can use to install new R packages on a server, especially one that does not have internet access. After creating the repository, you can modify it by adding new packages or upgrading the version of existing packages.

Package repositories are useful in these scenarios:

- **Security:** Many R users are accustomed to downloading and installing new R packages at will, from CRAN or one of its mirror sites. However, for security reasons, production servers running SQL Server typically do not have internet connectivity.
- **Easier offline installation:** To install a package to an offline server requires that you also download all package dependencies. Using miniCRAN makes it easier to get

all dependencies in the correct format and avoid dependency errors.

- **Improved version management:** In a multi-user environment, there are good reasons to avoid unrestricted installation of multiple package versions on the server. Use a local repository to provide a consistent set of packages for your users.

Install miniCRAN

The **miniCRAN** package itself is dependent on 18 other CRAN packages, among which is the **RCurl** package, which has a system dependency on the **curl-devel** package. Similarly, package **XML** has a dependency on **libxml2-devel**. To resolve dependencies, we recommend that you build your local repository initially on a machine with full internet access.

Run the following commands on a computer with a base R, R tools, and internet connection. It's assumed that this is not your SQL Server computer. The following commands install the **miniCRAN** package and the **igraph** package. This example checks whether the package is already installed, but you can bypass the `if` statements and install the packages directly.

```
R
```

```
if(!require("miniCRAN")) install.packages("miniCRAN")
if(!require("igraph")) install.packages("igraph")
library("miniCRAN")
```

Set the CRAN mirror and MRAN snapshot

Specify a mirror site to use in getting packages. For example, you could use the MRAN site, or any other site in your region that contains the packages you need. If a download fails, try [another mirror site](#).

```
R
```

```
CRAN_mirror <- c(CRAN = "https://mirrors.nics.utk.edu/cran/")
```

Create a local folder

Create a local folder in which to store the collected packages. If you repeat this often, you might want to use a descriptive name, such as "miniCRANZooPackages" or "miniCRANMyRPackageV2".

Specify the folder as the local repo. R syntax uses a forward slash for path names, which is opposite from Windows conventions.

```
R
```

```
local_repo <- "C:/miniCRANZooPackages"
```

Add packages to the local repo

After **miniCRAN** is installed and loaded, create a list that specifies the additional packages you want to download.

Do **not** add dependencies to this initial list. The **igraph** package used by **miniCRAN** generates the list of dependencies automatically. For more information about how to use the generated dependency graph, see [Using miniCRAN to identify package dependencies](#).

1. Add target packages "zoo" and "forecast" to a variable.

```
R
```

```
pkgs_needed <- c("zoo", "forecast")
```

2. Optionally, plot the dependency graph. This is not necessary, but it can be informative.

```
R
```

```
plot(makeDepGraph(pkgs_needed))
```

3. Create the local repo. Be sure to change the R version, if necessary, to the version installed on your SQL Server instance. If you did a component upgrade, your version might be newer than the original version. For more information, see [Get R package information](#).

```
R
```

```
pkgs_expanded <- pkgDep(pkgs_needed, repos = CRAN_mirror);  
makeRepo(pkgs_expanded, path = local_repo, repos = CRAN_mirror, type =  
"win.binary", Rversion = "3.3");
```

From this information, the **miniCRAN** package creates the folder structure that you need to copy the packages to the SQL Server later.

At this point you should have a folder containing the packages you need and any additional packages that are required. The folder should contain a collection of zipped packages. Do not unzip the packages or rename any files.

Optionally, run the following code to list the packages contained in the local miniCRAN repository.

R

```
pdb <- as.data.frame(pkgAvail(local_repo, type = "win.binary", Rversion =  
"3.3"), stringsAsFactors = FALSE);  
head(pdb);  
pdb$Package;  
pdb[, c("Package", "Version", "License")]
```

Add packages to the instance library

After you have a local repository with the packages you need, move the package repository to the SQL Server computer. The following procedure describes how to install the packages using R tools.

1. Copy the folder containing the miniCRAN repository, in its entirety, to the server where you plan to install the packages. The folder typically has this structure:

```
<miniCRAN root>/bin/windows/contrib/version/<all packages>
```

In this procedure, we assume a folder off the root drive.

2. Open an R tool associated with the instance (for example, you could use Rgui.exe). Right-click and select **Run as administrator** to allow the tool to make updates to your system.

- For example, the file location for RGUI is `C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\bin\x64`.

3. Get the path for the instance library, and add it to the list of library paths.

For example,

R

```
outputlib <- "C:/Program Files/Microsoft SQL  
Server/MSSQL14.MSSQLSERVER/R_SERVICES/library"
```

- Specify the new location on the server where you copied the **miniCRAN** repository as `server_repo`.

In this example, we assume that you copied the repository to a temporary folder on the server.

```
R
```

```
inputlib <- "C:/miniCRANZooPackages"
```

- Since you're working in a new R workspace on the server, you must also furnish the list of packages to install.

```
R
```

```
mypackages <- c("zoo", "forecast")
```

- Install the packages, providing the path to the local copy of the miniCRAN repo.

```
R
```

```
install.packages(mypackages, repos = file.path("file://",  
normalizePath(inputlib, winslash = "/")), lib = outputlib, type =  
"win.binary", dependencies = TRUE);
```

- From the instance library, you can view the installed packages using a command like the following:

```
R
```


```
installed.packages()
```

Next steps

- [Get R package information](#)
- [R tutorials](#)

Tips for using R packages

Article • 08/01/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

This article provides helpful tips on using R packages in SQL Server. These tips are for DBAs who are unfamiliar with R, and experienced R developers who are unfamiliar with package access in a SQL Server instance.

If you're new to R

As an administrator installing R packages for the first time, knowing a few basics about R package management can help you get started.

Package dependencies

R packages frequently depend on multiple other packages, some of which might not be available in the default R library used by the instance. Sometimes a package requires a different version of a dependent package than what's already installed. Package dependencies are noted in a DESCRIPTION file embedded in the package, but are sometimes incomplete. You can use a package called [iGraph](#) to fully articulate the dependency graph.

If you need to install multiple packages, or want to ensure that everyone in your organization gets the correct package type and version, we recommend that you use the [miniCRAN](#) package to analyze the complete dependency chain. miniCRAN creates a local repository that can be shared among multiple users or computers.

Package sources, versions, and formats

There are multiple sources for R packages, such as [CRAN](#) and [Bioconductor](#). The official site for the R language (<https://www.r-project.org/>) lists many of these resources. Many packages are published to GitHub, where developers can obtain the source code.

R packages run on multiple computing platforms. Be sure that the versions you install are Windows binaries.

Know which library you're installing to and which packages are already installed

If you have previously modified the R environment on the computer, before installing anything ensure that the R environment variable `.libPath` uses just one path.

This path should point to the R_SERVICES folder for the instance. For more information, including how to determine which packages are already installed, see [Get R package information](#).

If you're new to SQL Server

As an R developer working on code executing on SQL Server, the security policies protecting the server constrain your ability to control the R environment. The following tips describe typical situations and provide suggestions for working in this environment.

R user libraries: not supported on SQL Server

R developers who need to install new R packages are accustomed to installing packages at will, using a private, user library whenever the default library is not available, or when the developer is not an administrator on the computer. For example, in a typical R development environment, the user would add the location of the package to the R environment variable `libPath`, or reference the full package path, like this:

```
R
```

```
library("c:/Users/<username>/R/win-library/packageName")
```

This does not work when running R solutions in SQL Server, because R packages must be installed to a specific default library that is associated with the instance. When a package is not available in the default library, you get this error when you try to call the package:

Error in library(XXX) : there is no package called 'package-name'

For information on how to install R packages in SQL Server, see [Install new R packages on SQL Server Machine Learning Services or SQL Server R Services](#).

How to avoid "package not found" errors

Using the following guidelines will help you avoid "package not found" errors.

- Eliminate dependencies on user libraries.

It's a bad development practice to install required R packages to a custom user library. This can lead to errors if a solution is run by another user who does not have access to the library location.

Also, if a package is installed in the default library, the R runtime loads the package from the default library, even if you specify a different version in the R code.


- Make sure your code is able to run in a shared environment.
- Avoid installing packages as part of a solution. If you don't have permissions to install packages, the code will fail. Even if you do have permissions to install packages, you should do so separately from other code that you want to execute.
- Check your code to make sure that there are no calls to uninstalled packages.
- Update your code to remove direct references to the paths of R packages or R libraries.
- Know which package library is associated with the instance. For more information, see [Get R package information](#).

See also

- [Install packages with R tools](#)

Monitor Python and R script execution using custom reports in SQL Server Management Studio

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

Use custom reports in [SQL Server Management Studio \(SSMS\)](#) to monitor the execution of external scripts (Python and R), resources used, diagnose problems, and tune performance in [SQL Server Machine Learning Services](#).

In these reports, you can view details such as:

- Active Python or R sessions
- Configuration settings for the instance
- Execution statistics for machine learning jobs
- Extended events for R Services
- Python or R packages installed on the current instance


This article explains how to install and use the custom reports provided for SQL Server Machine Learning Services.

For more information on reports in SQL Server Management Studio, see [Custom reports in Management Studio](#).

How to install the reports

The reports are designed using SQL Server Reporting Services, but can be used directly from SQL Server Management Studio. Reporting Services does not have to be installed on your SQL Server instance.

To use these reports, follow these steps:

1. Download the [SSMS Custom Reports](#)  for SQL Server Machine Learning Services from GitHub.
2. Copy the reports to Management Studio
 - a. Locate the custom reports folder used by SQL Server Management Studio. By default, custom reports are stored in this folder (where **user_name** is your Windows user name):

C:\Users\user_name\Documents\SQL Server Management Studio\Custom Reports

You can also specify a different folder, or create subfolders.

b. Copy the *.RDL files you downloaded to the custom reports folder.

3. Run the reports in Management Studio

a. In Management Studio, right-click the **Databases** node for the instance where you want to run the reports.

b. Click **Reports**, and then click **Custom Reports**.

c. In the **Open File** dialog box, locate the custom reports folder.

d. Select one of the RDL files you downloaded, and then click **Open**.

Reports

The [SSMS Custom Reports repository in GitHub](#) includes the following reports:


Report	Description
Active Sessions	Users who are currently connected to the SQL Server instance and running a Python or R script.
Configuration	Installation settings of Machine Learning Services and properties of the Python or R runtime.
Configure Instance	Configure Machine Learning Services.
Execution Statistics	Execution statistics of Machine Learning services. For example, you can get the total number of external scripts executions and number of parallel executions.
Extended Events	Extended events that are available to get more insights into external scripts execution.
Packages	List the R or Python packages installed on the SQL Server instance and their properties, such as version and name.
Resource Usage	View the CPU, Memory, IO consumption of SQL Server, and external scripts execution. You can also view the memory setting for external resource pools.

Next steps

- Monitor SQL Server Machine Learning Services using dynamic management views (DMVs)
- Monitor Python and R scripts with extended events in SQL Server Machine Learning Services

Monitor SQL Server Machine Learning Services using dynamic management views (DMVs)

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

Use dynamic management views (DMVs) to monitor the execution of external scripts (Python and R), resources used, diagnose problems, and tune performance in SQL Server Machine Learning Services.

In this article, you will find the DMVs that are specific for SQL Server Machine Learning Services. You will also find example queries that show:

- Settings and configuration options for machine learning
- Active sessions running external Python or R scripts
- Execution statistics for the external runtime for Python and R
- Performance counters for external scripts
- Memory usage for the OS, SQL Server, and external resource pools
- Memory configuration for SQL Server and external resource pools
- Resource Governor resource pools, including external resource pools
- Installed packages for Python and R

For more general information about DMVs, see [System Dynamic Management Views](#).

Tip

You can also use the custom reports to monitor SQL Server Machine Learning Services. For more information, see [Monitor machine learning using custom reports in Management Studio](#).

Dynamic management views

The following dynamic management views can be used when monitoring machine learning workloads in SQL Server. To query the DMVs, you need `VIEW SERVER STATE` permission on the instance.

Dynamic management view	Type	Description
-------------------------	------	-------------

Dynamic management view	Type	Description
sys.dm_external_script_requests	Execution	Returns a row for each active worker account that is running an external script.
sys.dm_external_script_execution_stats	Execution	Returns one row for each type of external script request.
sys.dm_os_performance_counters	Execution	Returns a row per performance counter maintained by the server. If you use the search condition <code>WHERE object_name LIKE '%External Scripts%'</code> , you can use this information to see how many scripts ran, which scripts were run using which authentication mode, or how many R or Python calls were issued on the instance overall.
sys.dm_resource_governor_external_resource_pools	Resource Governor	Returns information about the current external resource pool state in Resource Governor, the current configuration of resource pools, and resource pool statistics.
sys.dm_resource_governor_external_resource_pool_affinity	Resource Governor	Returns CPU affinity information about the current external resource pool configuration in Resource Governor. Returns one row per scheduler in SQL Server where each scheduler is mapped to an individual processor. Use this view to monitor the condition of a scheduler or to identify runaway tasks.

For information about monitoring SQL Server instances, see [Catalog Views](#) and [Resource Governor Related Dynamic Management Views](#).

Settings and configuration

View the Machine Learning Services installation setting and configuration options.

	IsMLServicesInstalled	ExternalScriptsEnabled	ImpliedAuthenticationEnabled	IsTcpEnabled
1	1	1	0	0

Run the query below to get this output. For more information on the views and functions used, see [sys.dm_server_registry](#), [sys.configurations](#), and [SERVERPROPERTY](#).

SQL

```
SELECT CAST(SERVERPROPERTY('IsAdvancedAnalyticsInstalled') AS INT) AS
IsMLServicesInstalled
, CAST(value_in_use AS INT) AS ExternalScriptsEnabled
, COALESCE(SIGN(SUSER_ID(CONCAT (
    CAST(SERVERPROPERTY('MachineName') AS NVARCHAR(128))
    , '\SQLRUserGroup'
    , CAST(serverproperty('InstanceName') AS NVARCHAR(128))
))), 0) AS ImpliedAuthenticationEnabled
, COALESCE((
    SELECT CAST(r.value_data AS INT)
    FROM sys.dm_server_registry AS r
    WHERE r.registry_key LIKE 'HKLM\Software\Microsoft\Microsoft SQL
Server\%\SuperSocketNetLib\Tcp'
    AND r.value_name = 'Enabled'
), - 1) AS IsTcpEnabled
FROM sys.configurations
WHERE name = 'external scripts enabled';
```

The query returns the following columns:

Column	Description
IsMLServicesInstalled	Returns 1 if SQL Server Machine Learning Services is installed for the instance. Otherwise, returns 0.
ExternalScriptsEnabled	Returns 1 if external scripts is enabled for the instance. Otherwise, returns 0.
ImpliedAuthenticationEnabled	Returns 1 if implied authentication is enabled. Otherwise, returns 0. The configuration for implied authentication is checked by verifying if a login exists for SQLRUserGroup.

Column	Description
IsTcpEnabled	Returns 1 if the TCP/IP protocol is enabled for the instance. Otherwise, returns 0. For more information, see Default SQL Server Network Protocol Configuration .

Active sessions

View the active sessions running external scripts.

session_id	blocking_session_id	status	database_name	login_name	wait_time	wait_type	last_wait_type	total_elapsed_time	cpu_time	reads	logical_reads	writes	language	degree_of_parallelism	external_user_name
1	63	suspended	master	EXAMPLESERVER\ExampleUser	9113182	EXTERNAL_SCRIPT_NETWORK_IO	EXTERNAL_SCRIPT_NETWORK_IO	9114600	6	19	26	0	R	1	MSSQLSERVER01
2	54	suspended	master	EXAMPLESERVER\ExampleUser	9109227	EXTERNAL_SCRIPT_NETWORK_IO	EXTERNAL_SCRIPT_NETWORK_IO	9109452	2	0	26	0	R	1	MSSQLSERVER01

Run the query below to get this output. For more information on the dynamic management views used, see [sys.dm_exec_requests](#), [sys.dm_external_script_requests](#), and [sys.dm_exec_sessions](#).

```
SQL

SELECT r.session_id, r.blocking_session_id, r.status, DB_NAME(s.database_id)
AS database_name
    , s.login_name, r.wait_time, r.wait_type, r.last_wait_type,
r.total_elapsed_time, r.cpu_time
    , r.reads, r.logical_reads, r.writes, er.language,
er.degree_of_parallelism, er.external_user_name
FROM sys.dm_exec_requests AS r
INNER JOIN sys.dm_external_script_requests AS er
ON r.external_script_request_id = er.external_script_request_id
INNER JOIN sys.dm_exec_sessions AS s
ON s.session_id = r.session_id;
```

The query returns the following columns:

Column	Description
session_id	Identifies the session associated with each active primary connection.
blocking_session_id	ID of the session that is blocking the request. If this column is NULL, the request is not blocked, or the session information of the blocking session is not available (or cannot be identified).
status	Status of the request.
database_name	Name of the current database for each session.
login_name	SQL Server login name under which the session is currently executing.

Column	Description
wait_time	If the request is currently blocked, this column returns the duration in milliseconds, of the current wait. Is not nullable.
wait_type	If the request is currently blocked, this column returns the type of wait. For information about types of waits, see sys.dm_os_wait_stats .
last_wait_type	If this request has previously been blocked, this column returns the type of the last wait.
total_elapsed_time	Total time elapsed in milliseconds since the request arrived.
cpu_time	CPU time in milliseconds that is used by the request.
reads	Number of reads performed by this request.
logical_reads	Number of logical reads that have been performed by the request.
writes	Number of writes performed by this request.
language	Keyword that represents a supported script language.
degree_of_parallelism	Number indicating the number of parallel processes that were created. This value might be different from the number of parallel processes that were requested.
external_user_name	The Windows worker account under which the script was executed.

Execution statistics

View the execution statistics for the external runtime for R and Python. Only statistics of RevoScaleR, revoscalepy, or microsoftml package functions are currently available.

	language	counter_name	counter_value
1	Python	script_executions	7
2	R	script_executions	129

Run the query below to get this output. For more information on the dynamic management view used, see [sys.dm_external_script_execution_stats](#). The query only returns functions that have been executed more than once.

SQL

```
SELECT language, counter_name, counter_value
FROM sys.dm_external_script_execution_stats
WHERE counter_value > 0
ORDER BY language, counter_name;
```

The query returns the following columns:

Column	Description
language	Name of the registered external script language.
counter_name	Name of a registered external script function.
counter_value	Total number of instances that the registered external script function has been called on the server. This value is cumulative, beginning with the time that the feature was installed on the instance, and cannot be reset.

Performance counters

View the performance counters related to the execution of external scripts.

	counter_name	cntr_value
1	Total Executions	42
2	Parallel Executions	0
3	Streaming Executions	0
4	SQL CC Executions	0
5	Implied Auth. Logins	0
6	Total Execution Time (ms)	10599
7	Execution Errors	0

Run the query below to get this output. For more information on the dynamic management view used, see [sys.dm_os_performance_counters](#).

SQL

```
SELECT counter_name, cntr_value
FROM sys.dm_os_performance_counters
WHERE object_name LIKE '%External Scripts%'
```

`sys.dm_os_performance_counters` outputs the following performance counters for external scripts:

Counter	Description
Total Executions	Number of external processes started by local or remote calls.
Parallel Executions	Number of times that a script included the <i>@parallel</i> specification and that SQL Server was able to generate and use a parallel query plan.
Streaming Executions	Number of times that the streaming feature has been invoked.

Counter	Description
SQL CC Executions	Number of external scripts run where the call was instantiated remotely and SQL Server was used as the compute context.
Implied Auth. Logins	Number of times that an ODBC loopback call was made using implied authentication; that is, the SQL Server executed the call on behalf of the user sending the script request.
Total Execution Time (ms)	Time elapsed between the call and completion of call.
Execution Errors	Number of times scripts reported errors. This count does not include R or Python errors.

Memory usage

View information about the memory used by the OS, SQL Server, and the external pools.

	physical_memory_kb	committed_kb	external_pool_peak_memory_kb
1	3658780	207680	386476

Run the query below to get this output. For more information on the dynamic management views used, see [sys.dm_resource_governor_external_resource_pools](#) and [sys.dm_os_sys_info](#).

```
SQL

SELECT physical_memory_kb, committed_kb
      , (SELECT SUM(peak_memory_kb)
        FROM sys.dm_resource_governor_external_resource_pools AS ep
        ) AS external_pool_peak_memory_kb
FROM sys.dm_os_sys_info;
```

The query returns the following columns:

Column	Description
physical_memory_kb	The total amount of physical memory on the machine.
committed_kb	The committed memory in kilobytes (KB) in the memory manager. Does not include reserved memory in the memory manager.
external_pool_peak_memory_kb	The sum of the maximum amount of memory used, in kilobytes, for all external resource pools.

Memory configuration

View information about the maximum memory configuration in percentage of SQL Server and external resource pools. If SQL Server is running with the default value of `max server memory (MB)`, it is considered as 100% of the OS memory.

	name	max_memory_percent
1	SQL Server	83.962413718
2	External Pool - default	20.000000000

Run the query below to get this output. For more information on the views used, see [sys.configurations](#) and [sys.dm_resource_governor_external_resource_pools](#).

SQL

```
SELECT 'SQL Server' AS name
      , CASE CAST(c.value AS BIGINT)
          WHEN 2147483647 THEN 100
          ELSE (SELECT CAST(c.value AS BIGINT) / (physical_memory_kb / 1024.0)
                * 100 FROM sys.dm_os_sys_info)
          END AS max_memory_percent
FROM sys.configurations AS c
WHERE c.name LIKE 'max server memory (MB)'
UNION ALL
SELECT CONCAT ('External Pool - ', ep.name) AS pool_name,
       ep.max_memory_percent
FROM sys.dm_resource_governor_external_resource_pools AS ep;
```

The query returns the following columns:

Column	Description
name	Name of the external resource pool or SQL Server.
max_memory_percent	The maximum memory that SQL Server or the external resource pool can use.

Resource pools

In [SQL Server Resource Governor](#), a [resource pool](#) represents a subset of the physical resources of an instance. You can specify limits on the amount of CPU, physical IO, and memory that incoming application requests, including execution of external scripts, can use within the resource pool. View the resource pools used for SQL Server and external scripts.

	pool_name	total_cpu_usage_ms	read_io_completed_total	write_io_completed_total
1	SQL Server - internal	602	194	65
2	SQL Server - default	1177	215	12
3	External Pool - default	259843750	36975	2371

Run the query below to get this output. For more information on the dynamic management views used, see [sys.dm_resource_governor_resource_pools](#) and [sys.dm_resource_governor_external_resource_pools](#).

SQL

```
SELECT CONCAT ('SQL Server - ', p.name) AS pool_name
      , p.total_cpu_usage_ms, p.read_io_completed_total,
      p.write_io_completed_total
FROM sys.dm_resource_governor_resource_pools AS p
UNION ALL
SELECT CONCAT ('External Pool - ', ep.name) AS pool_name
      , ep.total_cpu_user_ms, ep.read_io_count, ep.write_io_count
FROM sys.dm_resource_governor_external_resource_pools AS ep;
```

The query returns the following columns:

Column	Description
pool_name	Name of the resource pool. SQL Server resource pools are prefixed with <code>SQL Server</code> and external resource pools are prefixed with <code>External Pool</code> .
total_cpu_usage_hours	The cumulative CPU usage in milliseconds since the Resource Governor statistics were reset.
read_io_completed_total	The total read IOs completed since the Resource Governor statistics were reset.
write_io_completed_total	The total write IOs completed since the Resource Governor statistics were reset.

Installed packages

You can to view the R and Python packages that are installed in SQL Server Machine Learning Services by executing an R or Python script that outputs these.

Installed packages for R

View the R packages installed in SQL Server Machine Learning Services.

	Package	Version	Depends	License	LibPath
1	base	3.3.3	NULL	Part of R 3.3.3	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
2	boot	1.3-18	R (>= 3.0.0), graphics, stats	Unlimited	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
3	checkpoint	0.4.0	R (>= 3.0.0)	GPL-2	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
4	class	7.3-14	R (>= 3.0.0), stats, utils	GPL-2 GPL-3	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
5	cluster	2.0.5	R (>= 3.0.1)	GPL (>= 2)	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
6	codetools	0.2-15	R (>= 2.1)	GPL	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
7	CompatibilityAPI	1.1.0	R (>= 3.2.2)	file LICENSE	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
8	compiler	3.3.3	NULL	Part of R 3.3.3	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
9	curl	2.6	R (>= 3.0.0)	MIT + file LICENSE	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
10	datasets	3.3.3	NULL	Part of R 3.3.3	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
11	deployRserve	9.0.0	R (>= 1.5.0)	GPL version 2	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
12	doParallel	1.0.10	R (>= 2.14.0), foreach (>= 1.2.0), iterators (>= 1.0.0), parallel, utils	GPL-2	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
13	doRSR	10.0.0	R (>= 2.5.0), foreach (>= 1.2.0), iterators (>= 1.0.0), RevoScaleR (>= 2.0-0), utils, RevoUtils	file LICENSE	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
14	foreach	1.4.5	R (>= 2.5.0)	Apache License (== 2.0)	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library
15	foreign	0.8-67	R (>= 3.0.0)	GPL (>= 2)	C:/Program Files/Microsoft SQL Server/MSSQL14.MSSQLSERVER/R_SERVICES/library

Run the query below to get this output. The query use an R script to determine R packages installed with SQL Server.

SQL

```
EXECUTE sp_execute_external_script @language = N'R'
, @script = N'
OutputDataSet <- data.frame(installed.packages()[,c("Package", "Version",
"Depends", "License", "LibPath")]);'
WITH result sets((Package NVARCHAR(255), Version NVARCHAR(100), Depends
NVARCHAR(4000)
, License NVARCHAR(1000), LibPath NVARCHAR(2000));
```

The columns returned are:

Column	Description
Package	Name of the installed package.
Version	Version of the package.
Depends	Lists the package(s) that the installed package depends on.
License	License for the installed package.
LibPath	Directory where you can find the package.

Installed packages for Python

View the Python packages installed in SQL Server Machine Learning Services.

	Package	Version	Location
1	xlwt	1.2.0	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
2	xlsxwriter	0.9.6	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
3	xlrd	1.0.0	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
4	win-unicode-console	0.5	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
5	widgetsnbextension	2.0.0	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
6	wheel	0.29.0	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
7	werkzeug	0.12.1	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
8	wcwidth	0.1.7	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
9	unicodectsv	0.14.1	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
10	traitlets	4.3.2	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
11	tomado	4.4.2	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
12	toolz	0.8.2	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
13	testpath	0.3	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
14	tables	3.2.2	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
15	sympy	1.0	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
16	statsmodels	0.8.0	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
17	sqlparse	0.1.19	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages
18	sqlalchemy	1.1.9	c:\program files\microsoft sql server\mssql14.mssqlserver\python_services\lib\site-packages

Run the query below to get this output. The query use an Python script to determine the Python packages installed with SQL Server.

SQL

```
EXECUTE sp_execute_external_script @language = N'Python'
, @script = N'
import pkg_resources
import pandas
OutputDataSet = pandas.DataFrame(sorted([(i.key, i.version, i.location) for
i in pkg_resources.working_set]))'
WITH result sets((Package NVARCHAR(128), Version NVARCHAR(128), Location
NVARCHAR(1000)));
```

The columns returned are:



Column	Description
Package	Name of the installed package.
Version	Version of the package.
Location	Directory where you can find the package.

Next steps

- [Extended events for machine learning](#)
- [Resource Governor Related Dynamic Management Views](#)
- [System Dynamic Management Views](#)
- [Monitor machine learning using custom reports in Management Studio](#)

Monitor Python and R scripts with extended events in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

Learn how to use extended events to monitor and troubleshooting operations related to the SQL Server Machine Learning Services, SQL Server Launchpad, and Python or R jobs external scripts.

Extended events for SQL Server Machine Learning Services

To view a list of events related to SQL Server Machine Learning Services, run the following query from Azure Data Studio or SQL Server Management Studio.

SQL

```
SELECT o.name AS event_name, o.description
FROM sys.dm_xe_objects o
JOIN sys.dm_xe_packages p
ON o.package_guid = p.guid
WHERE o.object_type = 'event'
AND p.name = 'SQLSatellite';
```

For more information about how to use extended events, see [Extended Events Tools](#).

Additional events specific to Machine Learning Services

Additional extended events are available for components that are related to and used by SQL Server Machine Learning Services, such as the SQL Server Launchpad, and BXLServer, and the satellite process that starts the Python or R runtime. These additional extended events are fired from the external processes; therefore, they must be captured using an external utility.

For more information about how to do this, see the section, [Collecting events from external processes](#).

Table of extended events

Event	Description	Notes
connection_accept	Occurs when a new connection is accepted. This event serves to log all connection attempts.	
failed_launching	Launching failed.	Indicates an error.
satellite_abort_connection	Abort connection record	
satellite_abort_received	Fires when an abort message is received over a satellite connection.	
satellite_abort_sent	Fires when an abort message is sent over satellite connection.	
satellite_authentication_completion	Fires when authentication completes for a connection over TCP or Named pipe.	
satellite_authorization_completion	Fires when authorization completes for a connection over TCP or Named pipe.	

Event	Description	Notes
satellite_cleanup	Fires when satellite calls cleanup.	Fired only from external process. See instructions on collecting events from external processes.
satellite_data_chunk_sent	Fires when the satellite connection finishes sending a single data chunk.	The event reports the number of rows sent, the number of columns, the number of SNI packets used and time elapsed in milliseconds while sending the chunk. The information can help you understand how much time is spent passing different types of data, and how many packets are used.
satellite_data_receive_completion	Fires when all the required data by a query is received over the satellite connection.	Fired only from external process. See instructions on collecting events from external processes.
satellite_data_send_completion	Fires when all required data for a session is sent over the satellite connection.	
satellite_data_send_start	Fires when data transmission starts.	Data transmission starts just before the first data chunk is sent.
satellite_error	Used for tracing sql satellite error	
satellite_invalid_sized_message	Message's size is not valid	
satellite_message_coalesced	Used for tracing message coalescing at networking layer	

Event	Description	Notes
satellite_message_ring_buffer_record	message ring buffer record	
satellite_message_summary	summary information about messaging	
satellite_message_version_mismatch	Message's version field is not matched	
satellite_messaging	Used for tracing messaging event (bind, unbind, etc.)	
satellite_partial_message	Used for tracing partial message at networking layer	
satellite_schema_received	Fires when schema message is received and read by SQL.	
satellite_schema_sent	Fires when schema message is sent by the satellite.	Fired only from external process. See instructions on collecting events from external processes.
satellite_service_start_posted	Fires when service start message is posted to launchpad.	This tells Launchpad to start the external process, and contains an ID for the new session.
satellite_unexpected_message_received	Fires when an unexpected message is received.	Indicates an error.

Event	Description	Notes
stack_trace	Occurs when a memory dump of the process is requested.	Indicates an error.
trace_event	Used for tracing purposes	These events can contain SQL Server, Launchpad, and external process trace messages. This includes output to stdout and stderr from R.
launchpad_launch_start	Fires when launchpad starts launching a satellite.	Fired only from Launchpad. See instructions on collecting events from launchpad.exe.
launchpad_resume_sent	Fires when launchpad has launched the satellite and sent a resume message to SQL Server.	Fired only from Launchpad. See instructions on collecting events from launchpad.exe.
satellite_data_chunk_sent	Fires when the satellite connection finishes sending a single data chunk.	Contains information about the number of columns, number of rows, number of packets, and time elapsed sending the chunk.
satellite_sessionId_mismatch	Message's session ID is not expected	

Collecting events from external processes

SQL Server Machine Learning Services starts some services that run outside of the SQL Server process. To capture events related to these external processes, you must create an events trace configuration file and place the file in the same directory as the executable for the process.

- **SQL Server Launchpad**

To capture events related to the Launchpad, place the *.xml* file in the Binn directory for the SQL Server instance. In a default installation, this would be:

```
C:\Program Files\Microsoft SQL
Server\MSSQL_version_number.MSSQLSERVER\MSSQL\Binn.
```

- **BXLServer** is the satellite process that supports SQL extensibility with external script languages, such as R or Python. A separate instance of BxlServer is launched for each external language instance.

To capture events related to BXLServer, place the *.xml* file in the R or Python installation directory. In a default installation, this would be:

```
R: C:\Program Files\Microsoft SQL
Server\MSSQL_version_number.MSSQLSERVER\R_SERVICES\library\RevoScaleR\rxLibs\x
64.
```

```
Python: C:\Program Files\Microsoft SQL
Server\MSSQL_version_number.MSSQLSERVER\PYTHON_SERVICES\Lib\site-
packages\revoscalepy\rxLibs.
```

The configuration file must be named the same as the executable, using the format "[name].xevents.xml". In other words, the files must be named as follows:

- `Launchpad.xevents.xml`
- `bxlserver.xevents.xml`

The configuration file itself has the following format:

```
XML

<?xml version="1.0" encoding="utf-8"?>
<event_sessions>
<event_session name="[session name]" maxMemory="1" dispatchLatency="1"
MaxDispatchLatency="2 SECONDS">
  <description owner="you">Xevent for launchpad or bxl server.
</description>
  <event package="SQLSatellite" name="[XEvent Name 1]" />
  <event package="SQLSatellite" name="[XEvent Name 2]" />
  <target package="package0" name="event_file">
    <parameter name="filename" value="[SessionName].xel" />
    <parameter name="max_file_size" value="10" />
    <parameter name="max_rollover_files" value="10" />
  </target>
</event_session>
</event_sessions>
```

- To configure the trace, edit the *session name* placeholder, the placeholder for the filename ([SessionName].xel), and the names of the events you want to capture, For example, [XEvent Name 1], [XEvent Name 1]).
- Any number of event package tags may appear, and will be collected as long as the name attribute is correct.

Example: Capturing Launchpad events

The following example shows the definition of an event trace for the Launchpad service:

XML

```
<?xml version="1.0" encoding="utf-8"?>
<event_sessions>
<event_session name="sqlsatelliteut" maxMemory="1" dispatchLatency="1"
MaxDispatchLatency="2 SECONDS">
  <description owner="hay">Xevent for sql tdd runner.</description>
  <event package="SQLSatellite" name="launchpad_launch_start" />
  <event package="SQLSatellite" name="launchpad_resume_sent" />
  <target package="package0" name="event_file">
    <parameter name="filename" value="launchpad_session.xel" />
    <parameter name="max_file_size" value="10" />
    <parameter name="max_rollover_files" value="10" />
  </target>
</event_session>
</event_sessions>
```

- Place the .xml file in the Binn directory for the SQL Server instance.
- This file must be named Launchpad.xevents.xml.

Example: Capturing BXLServer events

The following example shows the definition of an event trace for the BXLServer executable.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<event_sessions>
  <event_session name="sqlsatelliteut" maxMemory="1" dispatchLatency="1"
MaxDispatchLatency="2 SECONDS">
    <description owner="hay">Xevent for sql tdd runner.</description>
    <event package="SQLSatellite" name="satellite_abort_received" />
    <event package="SQLSatellite" name="satellite_authentication_completion"
/>
    <event package="SQLSatellite" name="satellite_cleanup" />
    <event package="SQLSatellite" name="satellite_data_receive_completion"
```

```
</>
  <event package="SQLSatellite" name="satellite_data_send_completion" />
  <event package="SQLSatellite" name="satellite_data_send_start" />
  <event package="SQLSatellite" name="satellite_schema_sent" />
  <event package="SQLSatellite"
name="satellite_unexpected_message_received" />
  <event package="SQLSatellite" name="satellite_data_chunk_sent" />
  <target package="package0" name="event_file">
    <parameter name="filename" value="satellite_session.xel" />
    <parameter name="max_file_size" value="10" />
    <parameter name="max_rollover_files" value="10" />
  </target>
</event_session>
</event_sessions>
```

- Place the *.xml* file in the same directory as the BXLServer executable.
- This file must be named `bxlserver.xevents.xml`.

Next steps

- [Monitor Python and R script execution using custom reports in SQL Server Management Studio](#)
- [Monitor SQL Server Machine Learning Services using dynamic management views \(DMVs\)](#)

Monitor PREDICT T-SQL statements with extended events in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2017 (14.x) and later  [Azure SQL Managed Instance](#)

Learn how to use extended events to monitor and troubleshooting [PREDICT T-SQL](#) statements in SQL Server Machine Learning Services.

Table of extended events

The following extended events are available in all versions of SQL Server that support the [PREDICT T-SQL](#) statement.

name	object_type	description
predict_function_completed	event	Builtin execution time breakdown
predict_model_cache_hit	event	Occurs when a model is retrieved from the PREDICT function model cache. Use this event along with other predict_model_cache_* events to troubleshoot issues caused by the PREDICT function model cache.
predict_model_cache_insert	event	Occurs when a model is insert into the PREDICT function model cache. Use this event along with other predict_model_cache_* events to troubleshoot issues caused by the PREDICT function model cache.
predict_model_cache_miss	event	Occurs when a model is not found in the PREDICT function model cache. Frequent occurrences of this event could indicate that SQL Server needs more memory. Use this event along with other predict_model_cache_* events to troubleshoot issues caused by the PREDICT function model cache.
predict_model_cache_remove	event	Occurs when a model is removed from model cache for PREDICT function. Use this event along with other predict_model_cache_* events to troubleshoot issues caused by the PREDICT function model cache.

Query for related events

To view a list of all columns returned for these events, run the following query in SQL Server Management Studio:

SQL

```
SELECT *
FROM sys.dm_xe_object_columns
WHERE object_name LIKE 'predict%'
```

Examples

To capture information about performance of a scoring session using PREDICT:

1. Create a new extended event session, using Management Studio or another supported [tool](#).
2. Add the events `predict_function_completed` and `predict_model_cache_hit` to the session.
3. Start the extended event session.
4. Run the query that uses PREDICT.

In the results, review these columns:

- The value for `predict_function_completed` shows how much time the query spent on loading the model and scoring.
- The boolean value for `predict_model_cache_hit` indicates whether the query used a cached model or not.

Native scoring model cache

In addition to the events specific to PREDICT, you can use the following queries to get more information about the cached model and cache usage:

View the native scoring model cache:

SQL

```
SELECT *
FROM sys.dm_os_memory_clerks
WHERE type = 'CACHESTORE_NATIVESCORING';
```

View the objects in the model cache:

SQL

```
SELECT *  
FROM sys.dm_os_memory_objects  
WHERE TYPE = 'MEMOBJ_NATIVE SCORING';
```

Next steps

For more information about extended events (sometimes called XEvents), and how to track events in a session, see these articles:

- [Monitor Python and R scripts with extended events in SQL Server Machine Learning Services](#)
- [Extended Events concepts and architecture](#)
- [Set up event capture in SSMS](#)
- [Manage event sessions in the Object Explorer](#)

Scale concurrent execution of external scripts in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Learn about worker accounts for SQL Server Machine Learning Services and how to change the default configuration to scale the number of concurrent execution of external scripts.

As part of the installation process for Machine Learning Services, a new Windows *user account pool* is created to support execution of tasks by the SQL Server Launchpad service. The purpose of these worker accounts is to isolate concurrent execution of external scripts by different SQL Server users.

Note

In SQL Server 2019, **SQLRUserGroup** only has one member which is now the single SQL Server Launchpad service account instead of multiple worker accounts. This article describes the worker accounts for SQL Server 2016 and 2017.

Worker account group

A Windows account group is created by SQL Server setup for each instance on which machine learning is installed and enabled.

- In a default instance, the group name is **SQLRUserGroup**. The name is the same whether you use Python or R or both.
- In a named instance, the default group name is suffixed with the instance name: for example, **SQLRUserGroupMyInstanceName**.

By default, the user account pool contains 20 user accounts. In most cases, 20 is more than adequate to support machine learning tasks, but you can change the number of accounts. The maximum number of accounts is 100.

- In a default instance, the individual accounts are named **MSSQLSERVER01** through **MSSQLSERVER20**.

- For a named instance, the individual accounts are named after the instance name: for example, **MyInstanceName01** through **MyInstanceName20**.

If more than one instance uses machine learning, the computer will have multiple user groups. A group cannot be shared across instances.

Number of worker accounts

To modify the number of users in the account pool, you must edit the properties of the SQL Server Launchpad service as described below.

Passwords associated with each user account are generated at random, but you can change them later, after the accounts have been created.

1. Open SQL Server Configuration Manager and select **SQL Server Services**.
2. Double-click the SQL Server Launchpad service and stop the service if it is running.
3. On the **Service** tab, make sure that the Start Mode is set to Automatic. External scripts cannot start when the Launchpad is not running.
4. Click the **Advanced** tab and edit the value of **External Users Count** if necessary. This setting controls how many different SQL users can run external script sessions concurrently. The default is 20 accounts. The maximum number of users is 100.
5. Optionally, you can set the option **Reset External Users Password** to *Yes* if your organization has a policy that requires changing passwords on a regular basis. Doing this will regenerate the encrypted passwords that Launchpad maintains for the user accounts. For more information, see [Enforcing Password Policy](#).
6. Restart the Launchpad service.

Managing workloads

The number of accounts in this pool determines how many external script sessions can be active simultaneously. By default, 20 accounts are created, meaning that 20 different users can have active Python or R sessions at one time. You can increase the number of worker accounts, if you expect to run more than 20 concurrent scripts.

When the same user executes multiple external scripts concurrently, all the sessions run by that user use the same worker account. For example, a single user might have 100 different Python or R scripts running concurrently, as long as resources permit, but all scripts would run using a single worker account.

The number of worker accounts that you can support, and the number of concurrent sessions that any single user can run, is limited only by server resources. Typically,

memory is the first bottleneck that you will encounter when using the Python or R runtime.

The resources that can be used by Python or R scripts are governed by SQL Server. We recommend that you monitor resource usage using SQL Server DMVs, or look at performance counters on the associated Windows job object, and adjust server memory use accordingly. If you have SQL Server Enterprise Edition, you can allocate resources used for running external scripts by configuring an [external resource pool](#).

Next steps

- [Monitor Python and R script execution using custom reports in SQL Server Management Studio](#)
- [Monitor SQL Server Machine Learning Services using dynamic management views \(DMVs\)](#)

Manage Python and R workloads with Resource Governor in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Learn how to use [Resource Governor](#) to manage CPU, physical IO, and memory resources allocation for Python and R workloads in SQL Server Machine Learning Services.

Machine learning algorithms in Python and R are computed intensive. Depending on your workload priorities, you might need to increase or decrease the resources available for Machine Learning Services.

For more general information, see [Resource Governor](#).

Note

Resource Governor is an Enterprise Edition feature.

Default allocations

By default, the external script runtimes for machine learning are limited to no more than 20% of total machine memory. It depends on your system, but in general, you might find this limit inadequate for serious machine learning tasks such as training a model or predicting on many rows of data.

Manage resources with Resource Governor

By default, external processes use up to 20% of total host memory on the local server. You can modify the default resource pool to make server-wide changes, with R and Python processes using whatever capacity you make available to external processes.

Optionally, you can create custom **external resource pools**, with associated workload groups and classifiers, to determine resource allocation for requests originating from specific programs, hosts, or other criteria that you provide. An external resource pool is

a type of resource pool introduced in SQL Server 2016 (13.x) to help manage the R and Python processes external to the database engine.

1. [Enable resource governance](#) (it's off by default).
2. Run `CREATE EXTERNAL RESOURCE POOL` to create and configure the resource pool, followed by `ALTER RESOURCE GOVERNOR` to implement it.
3. Create a workload group for granular allocations, for example between training and scoring.
4. Create a classifier to intercept calls for external processing.
5. Execute queries and procedures using the objects you created.

For a walkthrough, see [Create a resource pool for SQL Server Machine Learning Services](#) for step-by-step instructions.

For an introduction to terminology and general concepts, see [Resource Governor Resource Pool](#).

Processes under resource governance

You can use an *external resource pool* to manage the resources used by the following executables on a database engine instance:

- Rterm.exe when called locally from SQL Server or called remotely with SQL Server as the remote compute context
- Python.exe when called locally from SQL Server or called remotely with SQL Server as the remote compute context
- BxlServer.exe and satellite processes
- Satellite processes launched by Launchpad, such as PythonLauncher.dll

Note

Direct management of the Launchpad service by using Resource Governor is not supported. Launchpad is a trusted service that can only host launchers provided by Microsoft. Trusted launchers are explicitly configured to avoid consuming excessive resources.

Next steps

- Create a resource pool for machine learning
- Resource Governor resource pools

Create a resource pool for SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Learn how you can create and use a resource pool for managing Python and R workloads in SQL Server Machine Learning Services.

The process includes multiple steps:

1. Review status of any existing resource pools. It's important that you understand what services are using existing resources.
2. Modify server resource pools.
3. Create a new resource pool for external processes.
4. Create a classification function to identify external script requests.
5. Verify that the new external resource pool is capturing R or Python jobs from the specified clients or accounts.

Review the status of existing resource pools

1. Use a statement such as the following to check the resources assigned to the default pool for the server.

SQL

```
SELECT * FROM sys.resource_governor_resource_pools WHERE name = 'default'
```

Sample results

pool_id	name	min_cpu_percent	max_cpu_percent	min_memory_percent	max_memory_percent	cap_cpu_percent	min_iops_per_volume
2	default	0	100	0	100	100	0

2. Check the resources assigned to the default **external** resource pool.

SQL

```
SELECT * FROM sys.resource_governor_external_resource_pools WHERE name = 'default'
```

Sample results

external_pool_id	name	max_cpu_percent	max_memory_percent	max_processes	version
2	default	100	20	0	2

3. Under these server default settings, the external runtime will probably have insufficient resources to complete most tasks. To improve resources, you must modify the server resource usage as follows:

- Reduce the maximum computer memory that can be used by the database engine.
- Increase the maximum computer memory that can be used by the external process.

Modify server resource usage

1. In Management Studio, run the following statement to limit SQL Server memory usage to **60%** of the value in the 'max server memory' setting.

SQL

```
ALTER RESOURCE POOL "default" WITH (max_memory_percent = 60);
```

2. Run the following statement to limit the use of memory by external processes to **40%** of total computer resources.

SQL

```
ALTER EXTERNAL RESOURCE POOL "default" WITH (max_memory_percent = 40);
```


3. To enforce these changes, you must reconfigure and restart Resource Governor as follows:

SQL

```
ALTER RESOURCE GOVERNOR RECONFIGURE;
```

Note

These are just suggested settings to start with; you should evaluate your machine learning tasks in light of other server processes to determine the correct balance for your environment and workload.

Create a user-defined external resource pool

1. All changes to the configuration of Resource Governor are enforced across the server as a whole. The changes affect workloads that use the default pools for the server, as well as workloads that use the external pools.

To provide more fine-grained control over which workloads should have precedence, you can create a new user-defined external resource pool. Define a classification function and assign it to the external resource pool. The **EXTERNAL** keyword is new.

Create a new *user-defined external resource pool*. In the following example, the pool is named **ds_ep**.

SQL

```
CREATE EXTERNAL RESOURCE POOL ds_ep WITH (max_memory_percent = 40);
```

2. Create a workload group named **ds_wg** to use in managing session requests. For SQL queries you'll use the default pool; for all external process queries will use the **ds_ep** pool.

SQL

```
CREATE WORKLOAD GROUP ds_wg WITH (importance = medium) USING "default", EXTERNAL "ds_ep";
```

Requests are assigned to the default group whenever the request can't be classified, or if there's any other classification failure.

For more information, see [Resource Governor Workload Group](#) and [CREATE WORKLOAD GROUP \(Transact-SQL\)](#).

Create a classification function for machine learning

A classification function examines incoming tasks. It determines whether the task is one that can be run using the current resource pool. Tasks that do not meet the criteria of the classification function are assigned back to the server's default resource pool.

1. Begin by specifying that a classifier function should be used by Resource Governor to determine resource pools. You can assign a **null** as a placeholder for the classifier function.

SQL

```
ALTER RESOURCE GOVERNOR WITH (classifier_function = NULL);  
ALTER RESOURCE GOVERNOR RECONFIGURE;
```

For more information, see [ALTER RESOURCE GOVERNOR \(Transact-SQL\)](#).

2. In the classifier function for each resource pool, define the type of statements or incoming requests that should be assigned to the resource pool.

For example, the following function returns the name of the schema assigned to the user-defined external resource pool if the application that sent the request is either 'Microsoft R Host', 'RStudio', or 'Mashup'; otherwise it returns the default resource pool.

SQL

```
USE master  
GO  
CREATE FUNCTION is_ds_apps()  
RETURNS sysname  
WITH schemabinding  
AS  
BEGIN
```

```

IF program_name() in ('Microsoft R Host', 'RStudio', 'Mashup') RETURN 'ds_wg';
RETURN 'default'
END;
GO

```

- When the function has been created, reconfigure the resource group to assign the new classifier function to the external resource group that you defined earlier.

```

SQL

ALTER RESOURCE GOVERNOR WITH (classifier_function = dbo.is_ds_apps);
ALTER RESOURCE GOVERNOR RECONFIGURE;
GO

```

Verify new resource pools and affinity

Check the server memory configuration and CPU for each of the workload groups. Verify the instance resource changes have been made, by reviewing:

- the default pool for the SQL Server server
- the default resource pool for external processes
- the user-defined pool for external processes

- Run the following statement to view all workload groups:

```

SQL

SELECT * FROM sys.resource_governor_workload_groups;

```

Sample results

group_id	name	importance	request_max_memory_grant_percent	request_max_cpu_time_sec	request_memory_grant_timeout_sec	max_dop
1	internal	Medium	25	0	0	0
2	default	Medium	25	0	0	0
256	ds_wg	Medium	25	0	0	0

- Use the new catalog view, [sys.resource_governor_external_resource_pools \(Transact-SQL\)](#), to view all external resource pools.

```

SQL

SELECT * FROM sys.resource_governor_external_resource_pools;

```

Sample results

external_pool_id	name	max_cpu_percent	max_memory_percent	max_processes	version
2	default	100	20	0	2
256	ds_ep	100	40	0	1

For more information, see [Resource Governor Catalog Views \(Transact-SQL\)](#).

- Run the following statement to return information about the computer resources that are affinityized to the external resource pool, if applicable:

```

SQL

SELECT * FROM sys.resource_governor_external_resource_pool_affinity;

```

No information will be displayed because the pools were created with an affinity of AUTO. For more information, see [sys.dm_resource_governor_resource_pool_affinity \(Transact-SQL\)](#).

Next steps

For more information about managing server resources, see:



- [Resource Governor](#)
- [Resource Governor Related Dynamic Management Views \(Transact-SQL\)](#)

For an overview of resource governance for machine learning, see:

- [Manage Python and R workloads with Resource Governor in SQL Server Machine Learning Services](#)

Grant database users permission to execute Python and R scripts with SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later  [Azure SQL Managed Instance](#)

Learn how you can give a [database user](#) permission to run external Python and R scripts in [SQL Server Machine Learning Services](#) and give read, write, or data definition language (DDL) permissions to databases.

For more information, see the permissions section in [Security overview for the extensibility framework](#).

Permission to run scripts

For each user who runs Python or R scripts with SQL Server Machine Learning Services, and who are not an administrator, you must grant them the permission to run external scripts in each database where the language is used.

To grant permission to a [database user](#) to execute external script, run the following script:

SQL

```
USE <database_name>  
GO  
GRANT EXECUTE ANY EXTERNAL SCRIPT TO [UserName]
```

Note

Permissions are not specific to the supported script language. In other words, there are not separate permission levels for R script versus Python script.

Grant database permissions

While a database user is running scripts, the database user might need to read data from other databases. The database user might also need to create new tables to store

results, and write data into tables.

For each database user account or SQL login that is running R or Python scripts, ensure that it has the appropriate permissions on the specific database:

- `db_datareader` to read data.
- `db_datawriter` to save objects to the database.
- `db_ddladmin` to create objects such as stored procedures or tables containing trained and serialized data.

For example, the following Transact-SQL statement gives the SQL login *MySQLLogin* the rights to run T-SQL queries in the *ML_Samples* database. To run this statement, the SQL login must already exist in the security context of the server. For more information, see [sp_addrolemember \(Transact-SQL\)](#).

SQL

```
USE ML_Samples
GO
EXEC sp_addrolemember 'db_datareader', 'MySQLLogin'
```

Next steps

For more information about the permissions included in each role, see [Database-level roles](#).

SQL Server Launchpad service configuration

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

The SQL Server Launchpad is a service that manages and executes external scripts, similar to the way that the full-text indexing and query service launches a separate host for processing full-text queries.

For more information, see the Launchpad sections in [Extensibility architecture in SQL Server Machine Learning Services](#) and [Security overview for the extensibility framework in SQL Server Machine Learning Services](#).

Account permissions

By default, SQL Server Launchpad is configured to run under NT `Service\MSSQLLaunchpad`, which is provisioned with all necessary permissions to run external scripts. Removing permissions from this account can result in Launchpad failing to start or to access the SQL Server instance where external scripts should be run.

If you modify the service account, be sure to use the [Local Security Policy console](#).

Permissions required for this account are listed in the following table.

Group policy setting	Constant name
Adjust memory quotas for a process	SeIncreaseQuotaPrivilege
Bypass traverse checking	SeChangeNotifyPrivilege
Log on as a service	SeServiceLogonRight
Replace a process-level token	SeAssignPrimaryTokenPrivilege

For more information about permissions required to run SQL Server services, see [Configure Windows Service Accounts and Permissions](#).

Configuration properties

Typically, there is no reason to modify service configuration. Properties that could be changed include the service account, the count of external processes (20 by default), or

the password reset policy for worker accounts.

1. Open [SQL Server Configuration Manager](#).
2. Under SQL Server Services, right-click SQL Server Launchpad and select **Properties**.
 - To change the service account, click the **Log On** tab.
 - To increase the number of users, click the **Advanced** tab and change the **Security Contexts Count**.

ⓘ Note

In early versions of SQL Server 2016 R Services, you could change some properties of the service by editing the R Services (In-Database) configuration file. This file is no longer used for changing configurations. SQL Server Configuration Manager is the right approach for changes to service configuration, such as the service account and number of users.

Debug settings

A few properties can only be changed by using the Launchpad's configuration file, which might be useful in limited cases, such as debugging. The configuration file is created during the SQL Server setup and by default is saved as a plain text file in `<instance path>\bin\rlauncher.config`.

You must be an administrator on the computer that is running SQL Server to make changes to this file. If you edit the file, we recommend that you make a backup copy before saving changes.

The following table lists the advanced settings for SQL Server, with the permissible values.

Setting name	Type	Description
JOB_CLEANUP_ON_EXIT	Integer	<p>This is an internal setting only - do not change this value.</p> <p>Specifies whether the temporary working folder created for each external runtime session should be cleaned up after the session is completed. This setting is useful for debugging.</p> <p>Supported values are 0 (Disabled) or 1 (Enabled).</p> <p>The default is 1, meaning log files are removed on exit.</p>

Setting name	Type	Description
TRACE_LEVEL	Integer	<p>Configures the trace verbosity level of MSSQLLAUNCHPAD for debugging purposes. This affects trace files in the path specified by the LOG_DIRECTORY setting.</p> <p>Supported values are: 1 (Error), 2 (Performance), 3 (Warning), 4 (Information).</p> <p>The default is 1, meaning output errors only.</p>

All settings take the form of a key-value pair, with each setting on a separate line. For example, to change the trace level, you would add the line `Default: TRACE_LEVEL=4`.

Enforcing password policy

If your organization has a policy that requires changing passwords on a regular basis, you may need to force the Launchpad service to regenerate the encrypted passwords that Launchpad maintains for its worker accounts.

To enable this setting and force password refresh, open the **Properties** pane for the Launchpad service in SQL Server Configuration Manager, click **Advanced**, and change **Reset External Users Password** to **Yes**. When you apply this change, the passwords will immediately be regenerated for all user accounts. To run an external script after this change, you must restart the Launchpad service, at which time it will read the newly generated passwords.

To reset passwords at regular intervals, you can either set this flag manually or use a script.

Next steps

- [Extensibility framework](#)
- [Security overview](#)

Firewall configuration for SQL Server Machine Learning Services

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This article lists firewall configuration considerations that the administrator or architect should bear in mind when using [SQL Server Machine Learning Services](#) .

Default firewall rules

By default, the SQL Server Setup disables outbound connections by creating firewall rules.

In SQL Server 2016 and 2017, these rules are based on local user accounts, where Setup created one outbound rule for **SQLRUserGroup** that denied network access to its members (each worker account was listed as a local principal subject to the rule. For more information about SQLRUserGroup, see [Security overview for the extensibility framework in SQL Server Machine Learning Services](#).

In SQL Server 2019, as part of the move to AppContainers, there are new firewall rules based on AppContainer SIDs: one for each of the 20 AppContainers created by SQL Server Setup. Naming conventions for the firewall rule name are **Block network access for AppContainer-00 in SQL Server instance MSSQLSERVER**, where 00 is the number of the AppContainer (00-20 by default), and MSSQLSERVER is the name of the SQL Server instance.

Note

If network calls are required, you can disable the outbound rules in Windows Firewall.

Restrict network access

In a default installation, a Windows firewall rule is used to block all outbound network access from external runtime processes. Firewall rules should be created to prevent the external runtime processes from downloading packages or from making other network calls that could potentially be malicious.

If you are using a different firewall program, you can also create rules to block outbound network connection for external runtimes, by setting rules for the local user accounts or for the group represented by the user account pool.


We strongly recommend that you turn on Windows Firewall (or another firewall of your choice) to prevent unrestricted network access by the R or Python runtimes.

Next steps

[Configure Windows firewall for in-bound connections](#)

Create a login for SQLRUserGroup

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Create a [login in SQL Server](#) for `SQLRUserGroup` when a [loop back connection](#) in your script specifies a *trusted connection*, and the identity used to execute an object contains your code is a Windows user account.

Trusted connections are those having `Trusted_Connection=True` in the connection string. When SQL Server receives a request specifying a trusted connection, it checks whether the identity of the current Windows user has a login. For external processes executing as a worker account (such as `MSSQLSERVER01` from `SQLRUserGroup`), the request fails because those accounts do not have a login by default.

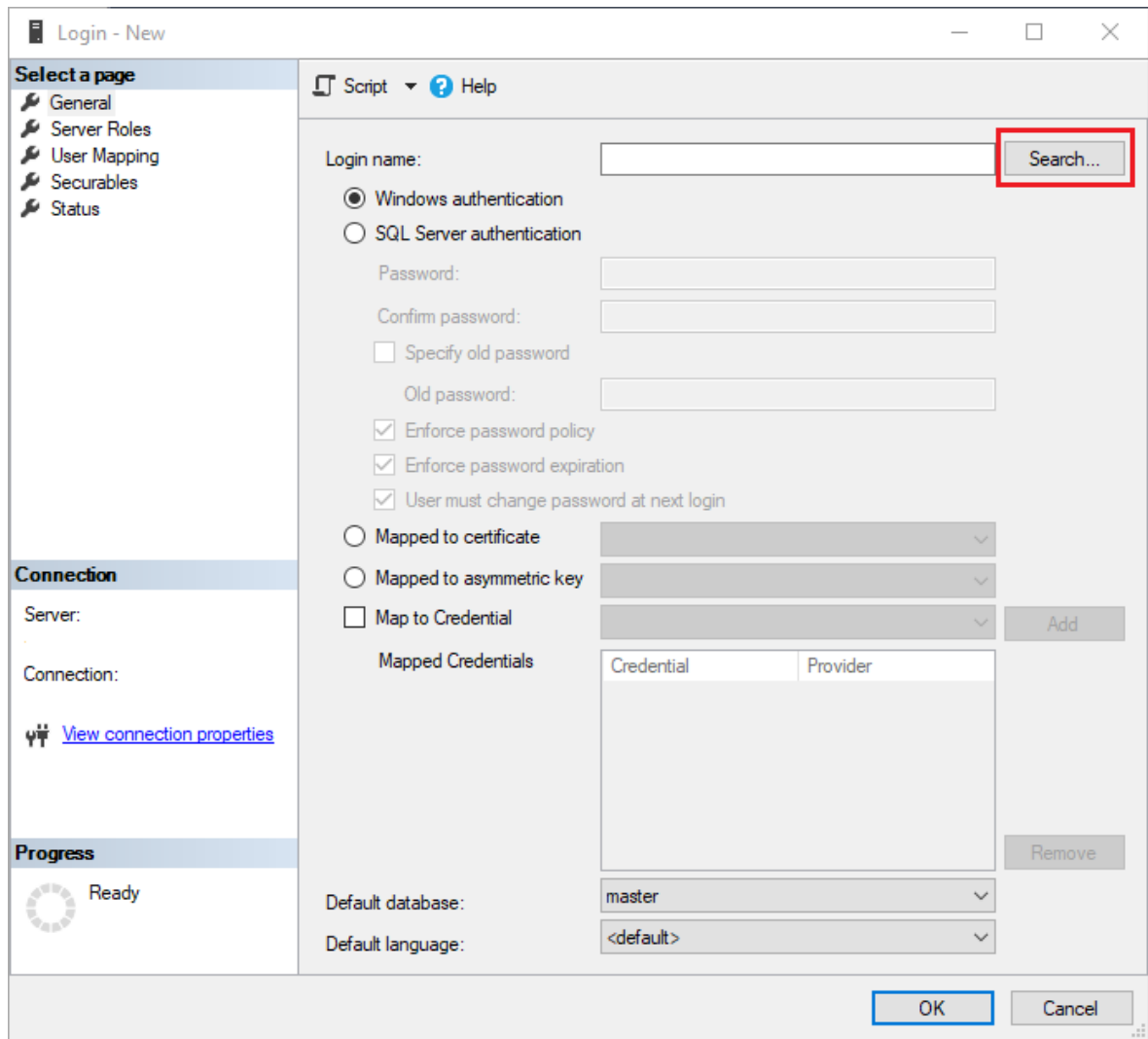
You can work around the connection error by creating a login for `SQLRUserGroup`. For more information about identities and external processes, see [Security overview for the extensibility framework](#).

Note

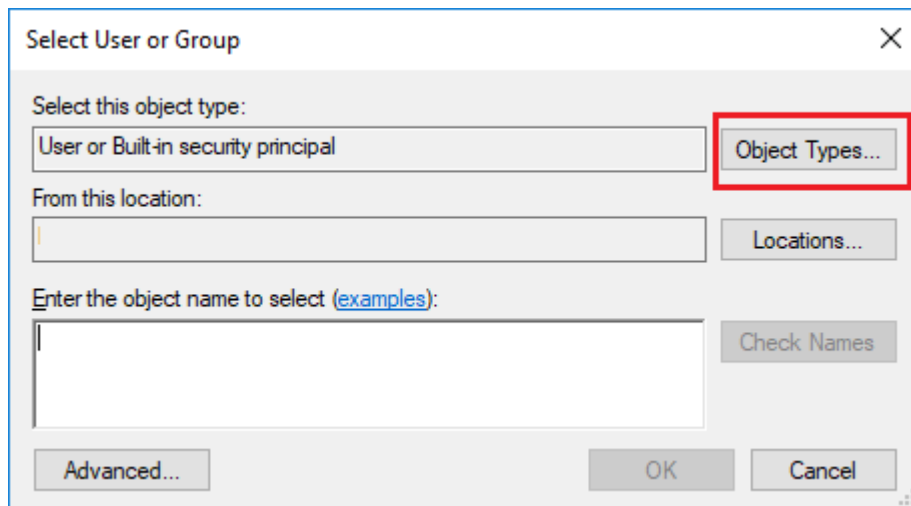
Make sure that `SQLRUserGroup` has "Allow Log on locally" permissions. By default, this right is given to all new local users, but some organizations stricter group policies might disable this right.

Create a login

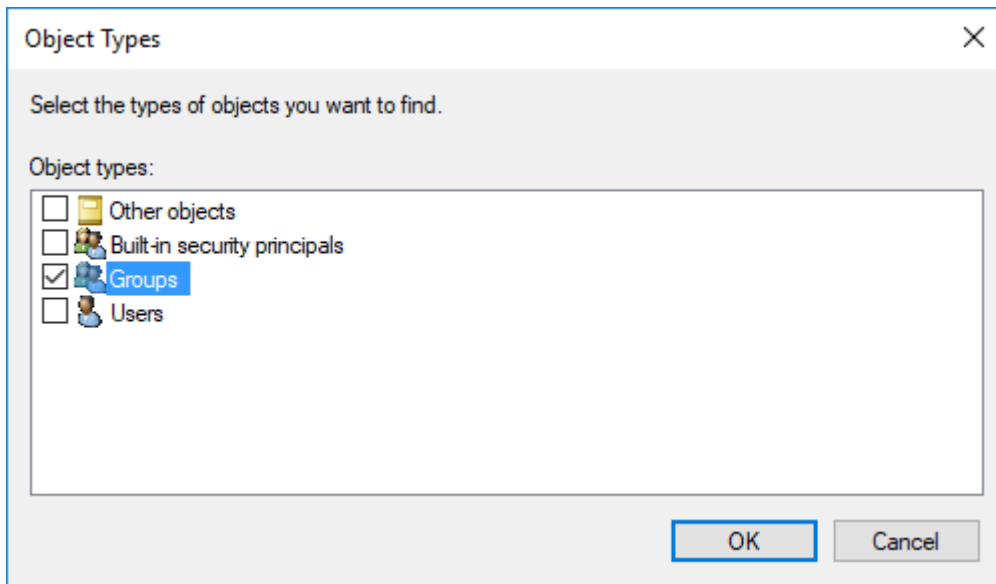
1. In SQL Server Management Studio, in Object Explorer, expand **Security**, right-click **Logins**, and select **New Login**.
2. In the **Login - New** dialog box, select **Search**. (Don't type anything in the box yet.)



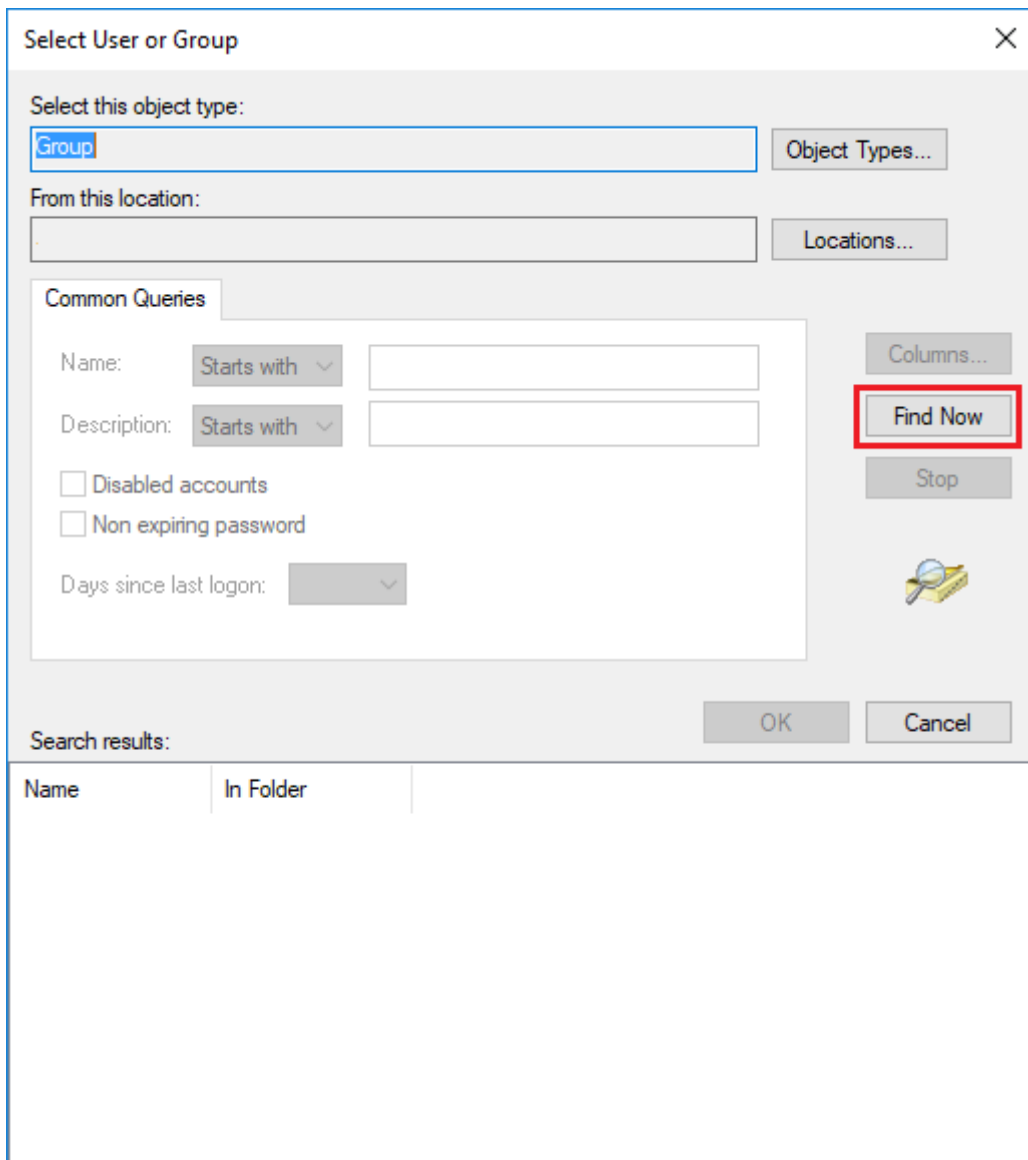
3. In the **Select User or Group** box, click the **Object Types** button.



4. In the **Object Types** dialog box, select **Groups**. Clear all other check boxes.

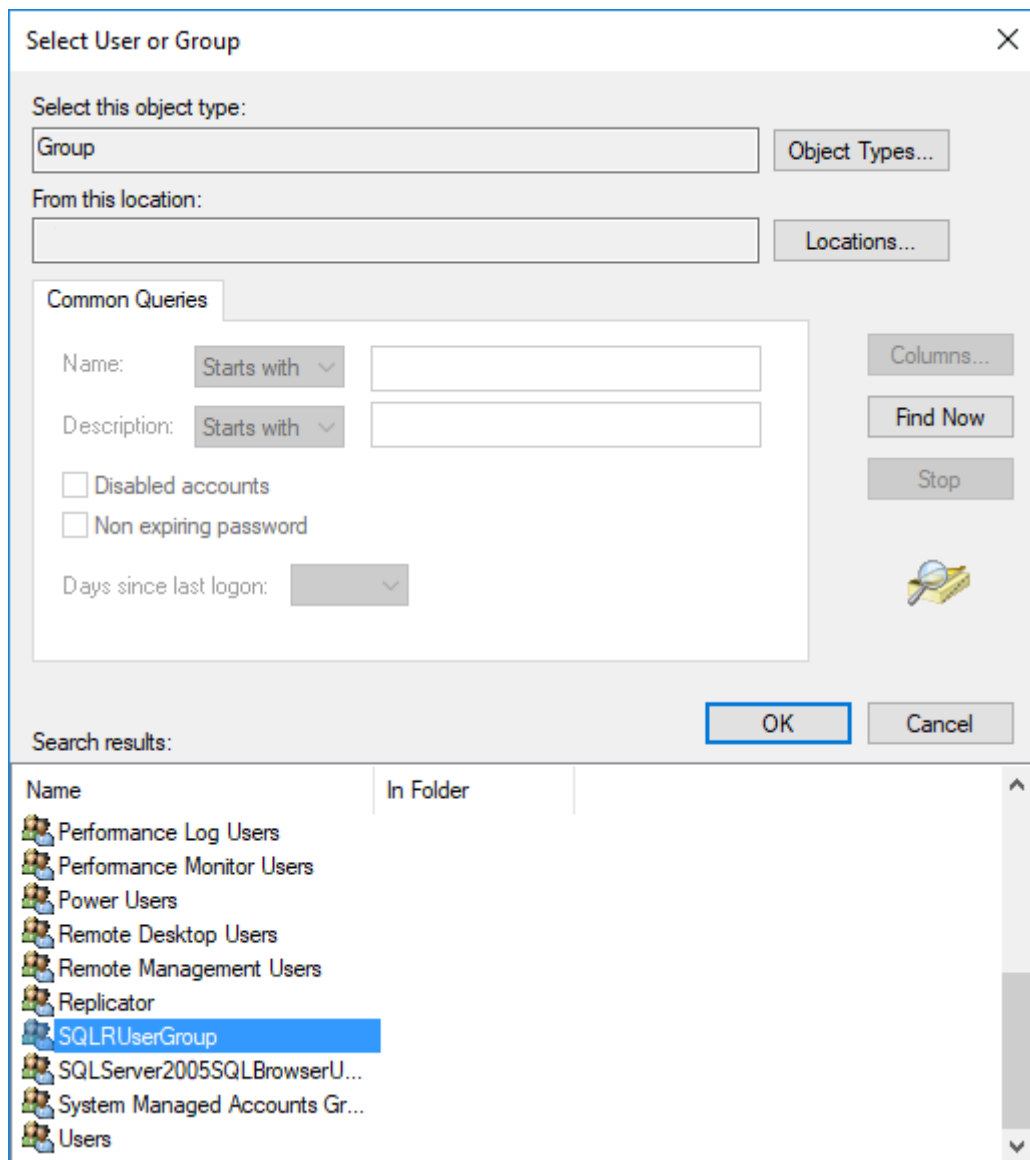


5. Click **Advanced**, verify that the location to search is the current computer, and then click **Find Now**.



6. Scroll through the list of group accounts on the server until you find one beginning with `SQLRUserGroup`.

- The name of the group that's associated with the Launchpad service for the *default instance* is always **SQLRUserGroup**, regardless of whether you installed R or Python or both. Select this account for the default instance only.
- If you are using a *named instance*, the instance name is appended to the name of the default worker group name, **SQLRUserGroup**. For example, if your instance is named "MLTEST", the default user group name for this instance would be **SQLRUserGroupMLTest**.



7. Click **OK** to close the advanced search dialog box.

i Important

Be sure you've selected the correct account for the instance. Each instance can use only its own Launchpad service and the group created for that service. Instances cannot share a Launchpad service or worker accounts.

8. Click **OK** once more to close the **Select User or Group** dialog box.
9. In the **Login - New** dialog box, click **OK**. By default, the login is assigned to the **public** role and has permission to connect to the database engine.

Next steps

- [Security overview](#)
- [Extensibility framework](#)

Performance tuning and data optimization for R

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This article discusses performance optimizations for R or Python scripts that run in SQL Server. You can use these methods to update your R code, both to boost performance and to avoid known issues.

Choosing a compute context

In SQL Server, you can use either the **local** or **SQL** compute context when running R or Python script.

When using the **local** compute context, analysis is performed on your computer and not on the server. Therefore, if you are getting data from SQL Server to use in your code, the data must be fetched over the network. The performance hit incurred for this network transfer depends on the size of the data transferred, speed of the network, and other network transfers occurring at the same time.

When using the **SQL Server compute context**, the code is executed on the server. If you are getting data from SQL Server, the data should be local to the server running the analysis, and therefore no network overhead is introduced. If you need to import data from other sources, consider arranging ETL beforehand.

When working with large data sets, you should always use the SQL compute context.

Factors

The R language has the concept of *factors*, which are special variable for categorical data. Data scientists often use factor variables in their formula, because handling categorical variables as factors ensures that the data is processed properly by machine learning functions.

By design, factor variables can be converted from strings to integers and back again for storage or processing. The R `data.frame` function handles all strings as factor variables, unless the argument `stringsAsFactors` is set to **False**. What this means is that strings are automatically converted to an integer for processing, and then mapped back to the original string.

If the source data for factors is stored as an integer, performance can suffer, because R converts the factor integers to strings at run time, and then performs its own internal string-to-integer conversion.

To avoid such run-time conversions, consider storing the values as integers in the SQL Server table, and using the *collInfo* argument to specify the levels for the column used as factor. Most data source objects in RevoScaleR take the parameter *collInfo*. You use this parameter to name the variables used by the data source, specify their type, and define the variables levels or transformations on the column values.

For example, the following R function call gets the integers 1, 2, and 3 from a table, but maps the values to a factor with levels "apple", "orange", and "banana".

R

```
c("fruit" = c(type = "factor", levels=as.character(c(1:3)),  
newLevels=c("apple", "orange", "banana")))
```

When the source column contains strings, it is always more efficient to specify the levels ahead of time using the *collInfo* parameter. For example, the following R code treats the strings as factors as they are being read.

R

```
c("fruit" = c(type = "factor", levels= c("apple", "orange", "banana")))
```

If there is no semantic difference in the model generation, then the latter approach can lead to better performance.

Data transformations

Data scientists often use transformation functions written in R as part of the analysis. The transformation function is applied to each row retrieved from the table. In SQL Server, such transformations are applied to all rows retrieved in a batch, which requires communication between the R interpreter and the analytics engine. To perform the transformation, the data moves from SQL to the analytics engine and then to the R interpreter process and back.

For this reason, using transformations as part of your R code can have a significant adverse effect on the performance of the algorithm, depending on the amount of data involved.

It is more efficient to have all necessary columns in the table or view before performing analysis, and avoid transformations during the computation. If it is not possible to add additional columns to existing tables, consider creating another table or view with the transformed columns and use an appropriate query to retrieve the data.

Batch row reads

If you use a SQL Server data source (`RxSqlServerData`) in your code, we recommend that you try using the parameter `rowsPerRead` to specify batch size. This parameter defines the number of rows that are queried and then sent to the external script for processing. At run time, the algorithm sees only the specified number of rows in each batch.

The ability to control the amount of data that is processed at a time can help you solve or avoid problems. For example, if your input dataset is very wide (has many columns), or if the dataset has a few large columns (such as free text), you can reduce the batch size to avoid paging data out of memory.

By default, the value of this parameter is set to 50000, to ensure decent performance even on machines with low memory. If the server has enough available memory, increasing this value to 500,000 or even a million can yield better performance, especially for large tables.

The benefits of increasing batch size become evident on a large data set, and in a task that can run on multiple processes. However, increasing this value does not always produce the best results. We recommend that you experiment with your data and algorithm to determine the optimal value.

Parallel processing

To improve the performance of `rx` analytic functions, you can leverage the ability of SQL Server to execute tasks in parallel using available cores on the server computer.

There are two ways to achieve parallelization with R in SQL Server:

- **Use `@parallel`.** When using the `sp_execute_external_script` stored procedure to run an R script, set the `@parallel` parameter to `1`. This is the best method if your R script does **not** use RevoScaleR functions, which have other mechanisms for processing. If your script uses RevoScaleR functions (generally prefixed with "rx"), parallel processing is performed automatically and you do not need to explicitly set `@parallel` to `1`.

If the R script can be parallelized, and if the SQL query can be parallelized, then the database engine creates multiple parallel processes. The maximum number of processes that can be created is equal to the **maximum degree of parallelism** (MAXDOP) setting for the instance. All processes then run the same script, but receive only a portion of the data.

Thus, this method is not useful with scripts that must see all the data, such as when training a model. However, it is useful when performing tasks such as batch prediction in parallel. For more information on using parallelism with `sp_execute_external_script`, see the **Advanced tips: parallel processing** section of [Using R Code in Transact-SQL](#).

- **Use `numTasks = 1`.** When using `rx` functions in a SQL Server compute context, set the value of the `numTasks` parameter to the number of processes that you would like to create. The number of processes created can never be more than **MAXDOP**; however, the actual number of processes created is determined by the database engine and may be less than you requested.

If the R script can be parallelized, and if the SQL query can be parallelized, then SQL Server creates multiple parallel processes when running the `rx` functions. The actual number of processes that are created depends on a variety of factors. These include resource governance, current usage of resources, other sessions, and the query execution plan for the query used with the R script.

Query parallelization

In Microsoft R, you can work with SQL Server data sources by defining your data as an `RxSqlServerData` data source object.

Creates a data source based on an entire table or view:

```
R
```

```
RxSqlServerData(table= "airline", connectionString = sqlConnString)
```

Creates a data source based on a SQL query:

```
R
```

```
RxSqlServerData(sqlQuery= "SELECT [ArrDelay],[CRSDepTime],[DayOfWeek] FROM  
airlineWithIndex WHERE rowNum <= 100000", connectionString = sqlConnString)
```

ⓘ Note

If a table is specified in the data source instead of a query, R Services uses internal heuristics to determine the necessary columns to fetch from the table; however, this approach is unlikely to result in parallel execution.

To ensure that the data can be analyzed in parallel, the query used to retrieve the data should be framed in such a way that the database engine can create a parallel query plan. If the code or algorithm uses large volumes of data, make sure that the query given to `RxSqlServerData` is optimized for parallel execution. A query that does not result in a parallel execution plan can result in a single process for computation.

If you need to work with large datasets, use Management Studio or another SQL query analyzer before you run your R code, to analyze the execution plan. Then, take any recommended steps to improve the performance of the query. For example, a missing index on a table can affect the time taken to execute a query. For more information, see [Monitor and Tune for Performance](#).

Another common mistake that can affect performance is that a query retrieves more columns than are required. For example, if a formula is based on only three columns, but your source table has 30 columns, you are moving data unnecessarily.

- Avoid using `SELECT *`!
- Take some time to review the columns in the dataset and identify only the ones needed for analysis
- Remove from your queries any columns that contain data types that are incompatible with R code, such as GUIDS and rowguids
- Check for unsupported date and time formats
- Rather than load a table, create a view that selects certain values or casts columns to avoid conversion errors

Optimizing the machine learning algorithm

This section provides miscellaneous tips and resources that are specific to RevoScaleR and other options in Microsoft R.

💡 Tip

A general discussion of R optimization is out of the scope of this article. However, if you need to make your code faster, we recommend the popular article, [The R Inferno](#) [↗]. It covers programming constructs in R and common pitfalls in vivid

language and detail, and provides many specific examples of R programming techniques.

Optimizations for RevoScaleR

Many RevoScaleR algorithms support parameters to control how the trained model is generated. While the accuracy and correctness of the model is important, the performance of the algorithm might be equally important. To get the right balance between accuracy and training time, you can modify parameters to increase the speed of computation, and in many cases, improve performance without reducing the accuracy or correctness.

- [rxDTree](#)

`rxDTree` supports the `maxDepth` parameter, which controls the depth of the decision tree. As `maxDepth` is increased, performance can degrade, so it is important to analyze the benefits of increasing the depth vs. hurting performance.

You can also control the balance between time complexity and prediction accuracy by adjusting parameters such as `maxNumBins`, `maxDepth`, `maxComplete`, and `maxSurrogate`. Increasing the depth to beyond 10 or 15 can make the computation very expensive.

- [rxLinMod](#)

Try using the `cube` argument if the first dependent variable in the formula is a factor variable.

When `cube` is set to `TRUE`, the regression is performed using a partitioned inverse, which might be faster and use less memory than standard regression computation. If the formula has a large number of variables, the performance gain can be significant.

- [rxLogit](#)

Use the `cube` argument if the first dependent variable is a factor variable.

When `cube` is set to `TRUE`, the algorithm uses a partitioned inverse, which might be faster and use less memory. If the formula has a large number of variables, the performance gain can be significant.

For more information on optimization of RevoScaleR, see these articles:

- Support article: [Performance tuning options for rxDForest and rxDTree](#) [↗]
- Methods for controlling model fit in a boosted tree model: [Estimating Models Using Stochastic Gradient Boosting](#)
- Overview of how RevoScaleR moves and processes data: [Write custom chunking algorithms in ScaleR](#)
- Programming model for RevoScaleR: [Managing threads in RevoScaleR](#)
- Function reference for [rxDForest](#)
- Function reference for [rxBTrees](#)

Use MicrosoftML

We also recommend that you look into the new **MicrosoftML** package, which provides scalable machine learning algorithms that can use the compute contexts and transformations provided by RevoScaleR.

- [Get started with MicrosoftML](#)
- [How to choose a MicrosoftML algorithm](#)

Next steps

- For R functions you can use to improve the performance of your R code, see [Use R code profiling functions to improve performance](#).
- For more complete information about performance tuning on SQL Server, see [Performance Center for SQL Server Database Engine and Azure SQL Database](#).

Use R code profiling functions to improve performance

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This article describes performance tools provided by R packages to get information about internal function calls. You can use this information to improve the performance of your code.

Tip

This article provides basic resources to get you started. For expert guidance, we recommend the *Performance* section in "[Advanced R](#)" by [Hadley Wickham](#).

Using RPROF

[rprof](#) is a function included in the base package [utils](#), which is loaded by default.

In general, the *rprof* function works by writing out the call stack to a file, at specified intervals. You can then use the [summaryRprof](#) function to process the output file. One advantage of *rprof* is that it performs sampling, thus lessening the performance load from monitoring.

To use R profiling in your code, you call this function and specify its parameters, including the name of the location of the log file that will be written. Profiling can be turned on and off in your code. The following syntax illustrates basic usage:

```
R

# Specify profiling output file.
varOutputFile <- "C:/TEMP/run001.log")
Rprof(varOutputFile)

# Turn off profiling
Rprof(NULL)

# Restart profiling
Rprof(append=TRUE)
```

Note

Using this function requires that Windows Perl be installed on the computer where code is run. Therefore, we recommend that you profile code during development in an R environment, and then deploy the debugged code to SQL Server.

R System Functions

The R language includes many base package functions for returning the contents of system variables. For example, as part of your R code, you might use `Sys.timezone` to get the current time zone, or `Sys.Time` to get the system time from R.

To get information about individual R system functions, type the function name as the argument to the R `help()` function from an R command prompt.

```
R
```

```
help("Sys.time")
```

Debugging and Profiling in R

The documentation for Microsoft R Open, which is installed by default, includes a manual on developing extensions for the R language that discusses [profiling and debugging](#) in detail.

Next steps

- For more information about optimizing R scripts in SQL Server, see [Performance tuning and data optimization for R](#).
- For more complete information about performance tuning on SQL Server, see [Performance Center for SQL Server Database Engine and Azure SQL Database](#).
- For more information on the utils package, see [The R Utils Package](#).
- For in-depth discussions of R programming, see ["Advanced R" by Hadley Wickham](#).

Machine Learning Server: manage web services with azureml-model-management-sdk

Article • 02/28/2023

Applies to: Machine Learning Server, SQL Server 2017

'azureml-model-management-sdk' is a custom Python package developed by Microsoft. This package provides the classes and functions to deploy and interact with analytic web services. These web services are backed by code block and scripts in Python or R.

This topic is a high-level description of package functionality. These classes and functions can be called directly. For syntax and other details, see the individual function help topics in the table of contents.

Package details	Information
Current version:	1.0.1b7
Built on:	Anaconda ↗ distribution of Python 3.5 ↗
Package distribution:	Machine Learning Server 9.x SQL Server 2017 Machine Learning Server (Standalone)

How to use this package

The `azureml-model-management-sdk` package is installed as part of Machine Learning Server and SQL Server 2017 Machine Learning Server (Standalone) when you add Python to your installation. It is also [available locally on Windows](#). When you install these products, you get the full collection of proprietary packages plus a Python distribution with its modules and interpreters.

You can use any Python IDE to write Python scripts that call the classes and functions in `azureml-model-management-sdk`. However, the script must run on a computer having Machine Learning Server or SQL Server 2017 Machine Learning Server (Standalone) with Python.

Use cases

There are three primary use cases for this release:

- Adding authentication logic to your Python script
- Deploying standard or real-time Python web services
- Managing and consuming these web services

Main classes and functions

- [DeployClient](#)
- [MLServer](#)
- [Operationalization](#)
- [OperationalizationDefinition](#)
- [ServiceDefinition](#)
- [RealtimeDefinition](#)
- [Service](#)
- [ServiceResponse](#)
- [Batch](#)
- [BatchResponse](#)

Next steps

Add both Python modules to your computer by running setup:

- Set up [Machine Learning Server](#) for Python or [Python Machine Learning Services](#).

Next, follow this quickstart to try it yourself:

- [Quickstart: How to deploy Python model as a service](#)

Or, read this how-to article:

- [How to publish and manage web services in Python](#)

See also

- [Library Reference](#)

- [Install Machine Learning Server](#)
- [Install the Python interpreter and libraries on Windows](#)
- [How to authenticate in Python with this package](#)
- [How to list, get, and consume services in Python with this package](#)

Class DeployClient

Article • 02/28/2023

```
azureml.deploy.DeployClient(host, auth=None, use=None)
```

Defines the factory for creating Deployment Clients.

Basic Usage Module implementation plugin with `use` property:

Find and Load *module* from an import reference:

```
from azureml.deploy import DeployClient
from azureml.deploy.server import MLServer

host = 'http://localhost:12800'
ctx = ('username', 'password')
mls_client = DeployClient(host, use=MLServer, auth=ctx)
```

Find and Load *module* as defined by *use* from namespace str:

```
host = 'http://localhost:12800'
ctx = ('username', 'password')

mls_client = DeployClient(host, use=MLServer, auth=ctx)
mls_client = DeployClient(host, use='azureml.deploy.server.MLServer',
auth=ctx)
```

Find and Load *module* from a file/path tuple:

```
host = 'http://localhost:12800'
ctx = ('username', 'password')

use = ('azureml.deploy.server.MLServer', '/path/to/mlserver.py')
mls_client = DeployClient(host, use=use, auth=ctx)
```

Create a new Deployment Client.

Arguments

host

Server HTTP/HTTPS endpoint, including the port number.

auth

(optional) Authentication context. Not all deployment clients require authentication. The *auth* is required for **MLServer**

use

(required) Deployment implementation to use (ex) *use='MLServer'* to use The ML Server.

Class MLServer

Article • 02/28/2023

MLServer

Python

```
azureml.deploy.server.MLServer
```

Bases: [azureml.deploy.operationalization.Operationalization](#)

This module provides a service implementation for the ML Server.

authentication

Python

```
authentication(context)
```

Override

Authentication lifecycle method called by the framework. Invokes the authentication entry-point for the class hierarchy.

ML Server supports two forms of authentication contexts:

- LDAP: tuple (*username, password*)
- Azure Active Directory (AAD): dict {...}
- access-token: str =4534535

Arguments

context

The authentication context: LDAP, Azure Active Directory (AAD), or existing *access-token* string.

HttpException

If an HTTP fault occurred calling the ML Server.

create_or_update_service_pool

Python

```
create_or_update_service_pool(name, version, initial_pool_size,  
max_pool_size, **opts)
```

Creates or updates the pool for the published web service, with given initial and maximum pool sizes on the ML Server by *name* and *version*.

Example:

Python

```
>>> client.create_or_update_service_pool(  
    'regression',  
    version = 'v1.0.0',  
    initial_pool_size = 1,  
    maximum_pool_size = 10)  
<Response [200]>  
>>>
```

Arguments

name

The unique web service name.

version

The web service version.

initial_pool_size

The initial pool size for the web service.

max_pool_size

The max pool size for the web service. This cannot be less than `initial_pool_size`.

Returns

requests.models.Response: HTTP Status indicating if the request was submitted successfully or not.

HttpException

If an HTTP fault occurred calling the ML Server.

delete_service

Python

```
delete_service(name, **opts)
```

Delete a web service.

Python

```
success = client.delete_service('example', version='v1.0.1')
print(success)
True
```

Arguments

name

The web service name.

opts

The web service version (*version='v1.0.1'*).

Returns

A `bool` indicating the service deletion was succeeded.

HttpException

If an HTTP fault occurred calling the ML Server.

delete_service_pool

Python

```
delete_service_pool(name, version, **opts)
```

Delete the pool for the published web service on the ML Server by *name* and *version*.

Example:

Python

```
>>> client.delete_service_pool('regression', version = 'v1.0.0')
<Response [200]>
>>>
```

Arguments

name

The unique web service name.

version

The web service version.

Returns

requests.models.Response: HTTP Status if the pool was deleted for the service.

HttpException

If an HTTP fault occurred calling the ML Server.

deploy_realtime

Python

```
deploy_realtime(name, **opts)
```

Publish a new *real-time* web service on the ML Server by *name* and *version*.

All input and output types are defined as a `pandas.DataFrame`.

Example:

Python

```
model = rx_serialize_model(model, realtime_scoring_only=True)
opts = {
    'version': 'v1.0.0',
    'description': 'Real-time service description.',
    'serialized_model': model
}

service = client.deploy_realtime('scoring', **opts)
df = movie_reviews.as_df()
res = service.consume(df)
answer = res.outputs
```

ⓘ Note

Using `deploy_realtime()` in this fashion is identical to publishing a service using the fluent APIS `deploy()`

Arguments

name

The web service name.

opts

The service properties to publish as a `dict`. The *opts* supports the following optional properties:

- `version` (str) - Defines a unique alphanumeric web service version. If the version is left blank, a unique *guid* is generated in its place. Useful during service development before the author is ready to officially publish a semantic version to share.
- `description` (str) - The service description.
- `alias` (str) - The consume function name. Defaults to *consume*.

Returns

A new instance of [Service](#) representing the real-time service *redeployed*.

HttpException

If an HTTP fault occurred calling the ML Server.

deploy_service

Python

```
deploy_service(name, **opts)
```

Publish a new web service on the ML Server by *name* and *version*.

Example:

Python

```
opts = {
    'version': 'v1.0.0',
    'description': 'Service description.',
    'code_fn': run,
    'init_fn': init,
    'objects': {'local_obj': 50},
    'models': {'model': 100},
    'inputs': {'x': int},
    'outputs': {'answer': float},
    'artifacts': ['histogram.png'],
    'alias': 'consume_service_fn_alias'
}

service = client.deploy('regression', **opts)
res = service.consume_service_fn_alias(100)
answer = res.output('answer')
histogram = res.artifact('histogram.png')
```

ⓘ Note

Using `deploy_service()` in this fashion is identical to publishing a service using the fluent APIS `deploy()`.

Arguments

name

The unique web service name.

opts

The service properties to publish. *opts* dict supports the following optional properties:

- version (str) - Defines a unique alphanumeric web service version. If the version is left blank, a unique *guid* is generated in its place. Useful during service development before the author is ready to officially publish a semantic version to share.
- description (str) - The service description.
- code_str (str) - A block of python code to run and evaluate.
- init_str (str) - A block of python code to initialize service.
- code_fn (function) - A Function to run and evaluate.
- init_fn (function) - A Function to initialize service.
- objects (dict) - Name and value of *objects* to include.
- models (dict) - Name and value of *models* to include.
- inputs (dict) - Service input schema by *name* and *type*. The following types are supported:
 - int
 - float
 - str
 - bool
 - numpy.array
 - numpy.matrix
 - pandas.DataFrame
- outputs (dict) - Defines the web service output schema. If empty, the service will not return a response value. *outputs* are defined as a dictionary `{'x'=int}` or `{'x':`

'int'} that describes the output parameter names and their corresponding data types. The following types are supported:

- int
- float
- str
- bool
- numpy.array
- numpy.matrix
- pandas.DataFrame
- artifacts (list) - A collection of file artifacts to return. File content is encoded as a *Base64 String*.
- alias (str) - The consume function name. Defaults to *consume*. If *code_fn* function is provided, then it will use that function name by default.

Returns

A new instance of [Service](#) representing the service *deployed*.

HttpException

If an HTTP fault occurred calling the ML Server.

destructor

```
Python
```

```
destructor()
```

Override

Destroy lifecycle method called by the framework. Invokes destructors for the class hierarchy.

get_service

```
Python
```

```
get_service(name, **opts)
```

Get a web service for consumption.

```
Python
```

```
service = client.get_service('example', version='v1.0.1')
print(service)
<ExampleService>
...
...
...
```

Arguments

name

The web service name.

opts

The optional web service version. If `version=None` the most recent service will be returned.

Returns

A new instance of [Service](#).

HttpException

If an HTTP fault occurred calling the ML Server.

get_service_pool_status

```
Python
```

```
get_service_pool_status(name, version, **opts)
```

Get status of pool on each compute node of the ML Server for the published services with the provided *name* and *version*.

Example:

Python

```
>>> client.create_or_update_service_pool(
    'regression',
    version = 'v1.0.0',
    initial_pool_size = 5,
    maximum_pool_size = 5)
<Response [200]>
>>> client.get_service_pool_status('regression', version = 'v1.0.0')
[{'computeNodeEndpoint': 'http://localhost:12805/', 'status': 'Pending'}]
>>> client.get_service_pool_status('regression', version = 'v1.0.0')
[{'computeNodeEndpoint': 'http://localhost:12805/', 'status': 'Success'}]
```

Arguments

name

The unique web service name.

version

The web service version.

Returns

str: json representing the status of pool on each compute node for the deployed service.

HttpException

If an HTTP fault occurred calling the ML Server.

Python

```
initializer(http_client, config, adapters=None)
```

Override

Init lifecycle method called by the framework, invoked during construction. Sets up attributes and invokes initializers for the class hierarchy.

Arguments

http_client

The http request session to manage and persist settings across requests (auth, proxies).

config

The global configuration.

adapters

A `dict` of transport adapters by url.

list_services

```
Python
```

```
list_services(name=None, **opts)
```

List the different published web services on the ML Server.

The service *name* and service *version* are optional. This call allows you to retrieve service information regarding:

- All services published
- All versioned services for a specific named service
- A specific version for a named service

Users can use this information along with the `[get_service()](#getservice)` operation to interact with and consume the web service.

Example:

```
Python
```

```
all_services = client.list_services()  
all_versions_of_add_service = client.list_services('add-service')
```



```
add_service_v1 = client.list_services('add-service', version='v1')
```

Arguments

name

The web service name.

opts

The optional web service version.

Returns

A `list` of service metadata.

HttpException

If an HTTP fault occurred calling the ML Server.

realtime_service

Python

```
realtime_service(name)
```

Begin fluent API chaining of properties for defining a *real-time* web service.

Example:

Python

```
client.realtime_service('scoring')  
    .description('A new real-time web service')  
    .version('v1.0.0')
```

Arguments

name

The web service name.

Returns

A [RealtimeDefinition](#) instance for fluent API chaining.

redeploy_realtime

Python

```
redeploy_realtime(name, **opts)
```

Updates properties on an existing *real-time* web service on the Server by *name* and *version*. If `version=None` the most recent service will be updated.

All input and output types are defined as a `pandas.DataFrame`.

Example:

Python

```
model = rx_serialize_model(model, realtime_scoring_only=True)
opts = {
    'version': 'v1.0.0',
    'description': 'Real-time service description.',
    'serialized_model': model
}

service = client.redeploy_realtime('scoring', **opts)
df = movie_reviews.as_df()
res = service.consume(df)
answer = res.outputs
```

ⓘ Note

Using `redeploy_realtime()` in this fashion is identical to updating a service using the fluent APIS `redeploy()`

Arguments

name

The web service name.

opts

The service properties to update as a `dict`. The *opts* supports the following optional properties:

- version (str) - Defines the web service version.
- description (str) - The service description.
- alias (str) - The consume function name. Defaults to *consume*.

Returns

A new instance of [Service](#) representing the real-time service *redeployed*.

HttpException

If an HTTP fault occurred calling the ML Server.

redeploy_service

Python

```
redeploy_service(name, **opts)
```

Updates properties on an existing web service on the ML Server by *name* and *version*. If `version=None` the most recent service will be updated.

Example:

Python

```
opts = {
    'version': 'v1.0.0',
    'description': 'Service description.',
    'code_fn': run,
    'init_fn': init,
    'objects': {'local_obj': 50},
    'models': {'model': 100},
    'inputs': {'x': int},
    'outputs': {'answer': float},
    'artifacts': ['histogram.png'],
    'alias': 'consume_service_fn_alias'
```

```
}  
  
service = client.redeploy('regression', **opts)  
res = service.consume_service_fn_alias(100)  
answer = res.output('answer')  
histogram = res.artifact('histogram.png')
```

ⓘ Note

Using `redeploy_service()` in this fashion is identical to updating a service using the fluent APIS `redeploy()`

Arguments

name

The web service name.

opts

The service properties to update as a `dict`. The `opts` supports the following optional properties:

- `version` (str) - Defines a unique alphanumeric web service version. If the version is left blank, a unique *guid* is generated in its place. Useful during service development before the author is ready to officially publish a semantic version to share.
- `description` (str) - The service description.
- `code_str` (str) - A block of python code to run and evaluate.
- `init_str` (str) - A block of python code to initialize service.
- `code_fn` (function) - A Function to run and evaluate.
- `init_fn` (function) - A Function to initialize service.
- `objects` (dict) - Name and value of *objects* to include.
- `models` (dict) - Name and value of *models* to include.
- `inputs` (dict) - Service input schema by *name* and *type*. The following types are supported: - int - float - str - bool - numpy.array - numpy.matrix -

pandas.DataFrame

- outputs (dict) - Defines the web service output schema. If empty, the service will not return a response value. *outputs* are defined as a dictionary `{'x'=int}` or `{'x': 'int'}` that describes the output parameter names and their corresponding data *types*. The following types are supported: - int - float - str - bool - numpy.array - numpy.matrix - pandas.DataFrame
- artifacts (list) - A collection of file artifacts to return. File content is encoded as a *Base64 String*.
- alias (str) - The consume function name. Defaults to *consume*. If *code_fn* function is provided, then it will use that function name by default.

Returns

A new instance of [Service](#) representing the service *deployed*.

HttpException

If an HTTP fault occurred calling the ML Server.

service

Python

```
service(name)
```

Begin fluent API chaining of properties for defining a *standard* web service.

Example:

Python

```
client.service('scoring')  
    .description('A new web service')  
    .version('v1.0.0')
```

Arguments

name

The web service name.

Returns

A [ServiceDefinition](#) instance for fluent API chaining.

Class Operationalization

Article • 03/24/2023

Operationalization

```
azureml.deploy.operationalization.Operationalization
```

Operationalization is designed to be a low-level abstract foundation class from which other service operationalization attribute classes in the *mldeploy* package can be derived. It provides a standard template for creating attribute-based operationalization lifecycle phases providing a consistent *init()*, *del()* sequence that chains initialization (initializer), authentication (authentication), and destruction (destructor) methods for the class hierarchy.

authentication

```
Python
```

```
authentication(context)
```

Authentication lifecycle method. Invokes the authentication entry-point for the class hierarchy.

An optional *noop* method where subclass implementers MAY provide this method definition by overriding.

Sub-class should override and implement.

Arguments

context

The optional authentication context as defined in the implementing sub-class.

delete_service

Python

```
delete_service(name, **opts)
```

Sub-class should override and implement.

deploy_realtime

Python

```
deploy_realtime(name, **opts)
```

Sub-class should override and implement.

deploy_service

Python

```
deploy_service(name, **opts)
```

Sub-class should override and implement.

destructor

Python

```
destructor()
```

Destroy lifecycle method. Invokes destructors for the class hierarchy.

An optional *noop* method where subclass implementers MAY provide this method definition by overriding.

Sub-class should override and implement.

get_service

Python

```
get_service(name, **opts)
```


Retrieve service metadata from the name source and return a new service instance.

Sub-class should override and implement.

initializer

Python

```
initializer(api_client, config, adapters=None)
```

Init lifecycle method, invoked during construction. Sets up attributes and invokes initializers for the class hierarchy.

An optional *noop* method where subclass implementers MAY provide this method definition by overriding.

Sub-class should override and implement.

list_services

Python

```
list_services(name=None, **opts)
```

Sub-class should override and implement.

realtime_service

Python

```
realtime_service(name)
```

Begin fluent API chaining of properties for defining a *real-time* web service.

Example:

```
client.realtime_service('scoring')  
    .description('A new real-time web service')  
    .version('v1.0.0')
```

Arguments

name

The web service name.

Returns

A [RealtimeDefinition](#) instance for fluent API chaining.

```
redeploy_realtime(name, force=False, **opts)
```

Sub-class should override and implement.

redeploy_service

Python

```
redeploy_service(name, force=False, **opts)
```

Sub-class should override and implement.

service

Python

```
service(name)
```

Begin fluent API chaining of properties for defining a *standard* web service.

Example:

```
client.service('scoring')  
    .description('A new web service')  
    .version('v1.0.0')
```

Arguments

name

The web service name.

Returns

A [ServiceDefinition](#) instance for fluent API chaining.

Class OperationalizationDefinition

Article • 02/28/2023

OperationalizationDefinition

```
azureml.deploy.operationalization.OperationalizationDefinition(name, op,  
    defs_extent={})
```

Base abstract class defining a service's properties.

Create a new publish definition.

Arguments

name

The web service name.

op

A reference to the deploy client instance.

defs_extent

A mixin of subclass specific definitions.

Python

```
alias(alias)
```

Set the optional service function name alias to use in order to consume the service.

Example:

```
service = client.service('score-service').alias('score').deploy()
```

```
# `score()` is the function that will call the `score-service`  
result = service.score()
```

Arguments

alias

The service function name alias to use in order to consume the service.

Returns

Self [OperationalizationDefinition](#) for fluent API.

deploy

```
Python
```

```
deploy()
```

Bundle up the definition properties and publish the service.

To be implemented by subclasses.

Returns

A new instance of [Service](#) representing the service *deployed*.

description

```
Python
```

```
description(description)
```

Set the service's optional description.

Arguments

description

The description of the service.

Returns

Self [OperationalizationDefinition](#) for fluent API.

redeploy

Python

```
redeploy(force=False)
```

Bundle up the definition properties and update the service.

To be implemented by subclasses.

Returns

A new instance of [Service](#) representing the service *deployed*.

version

Python

```
version(version)
```

Set the service's optional version.

Arguments

version

The version of the service.

Returns

Self [OperationalizationDefinition](#) for fluent API.

ServiceDefinition

Article • 02/28/2023

Class ServiceDefinition

```
azureml.deploy.operationalization.ServiceDefinition(name, op)
```

Bases: [azureml.deploy.operationalization.OperationalizationDefinition](#)

Service class defining a *standard* service's properties for publishing.

Python

```
alias(alias)
```

Set the optional service function name alias to use in order to consume the service.

Example:

```
service = client.service('score-service').alias('score').deploy()  
  
# `score()` is the function that will call the `score-service`  
result = service.score()
```

Arguments

alias

The service function name alias to use in order to consume the service.

Returns

Self [OperationalizationDefinition](#) for fluent API.

artifact

```
Python
```

```
artifact(artifact)
```

Define a service's optional supported file artifact by name. A convenience to calling `.artifacts(['file.png'])` with a list of one.

Arguments

artifact

A single file artifact by name.

Returns

Self [OperationalizationDefinition](#) for fluent API chaining.

artifacts

```
Python
```

```
artifacts(artifacts)
```

Defines a service's optional supported file artifacts by name.

Arguments

artifacts

A `list` of file artifacts by name.

Returns

Self [OperationalizationDefinition](#) for fluent API chaining.

code_fn

```
Python
```



```
code_fn(code, init=None)
```

Set the service consume function as a function.

Example:

```
def init():  
    pass  
  
def score(df):  
    pass  
  
.code_fn(score, init)
```

Arguments

code

A function handle as a reference to run python code.

init

An optional function handle as a reference to initialize the service.

Returns

Self [OperationalizationDefinition](#) for fluent API chaining.

code_str

Python

```
code_str(code, init=None)
```

Set the service consume function as a block of python code as a `str`.

```
init = 'import pandas as pd'  
code = 'print(pd)'
```

```
.code_str(code, init)
```

Arguments

code

A block of python code as a `str`.

init

An optional block of python code as a `str` to initialize the service.

Returns

A [ServiceDefinition](#) for fluent API chaining.

deploy

```
Python
```

```
deploy()
```

Bundle up the definition properties and publish the service.

Returns

A new instance of [Service](#) representing the service *deployed*.

description

```
Python
```

```
description(description)
```

Set the service's optional description.

Arguments

description

The description of the service.

Returns

Self [OperationalizationDefinition](#) for fluent API.

inputs

Python

```
inputs(**inputs)
```

Defines a service's optional supported inputs by name and type.

Example:

```
.inputs(a=float, b=int, c=str, d=bool, e='pandas.DataFrame')
```

Arguments

inputs

The inputs by name and type.

Returns

Self [OperationalizationDefinition](#) for fluent API chaining.

models

Python

```
models(**models)
```

Include any model(s) used for this service.

Example:

```
cars_model = rx_lin_mod(formula="am ~ hp + wt", data=mtcars)

.models(cars_model=cars_model)
```

Arguments

models

Any models by name and value.

Returns

Self [OperationalizationDefinition](#) for fluent API chaining.

objects

Python

```
objects(**objects)
```

Include any object(s) used for this service.

Example:

```
x = 5
y = 'hello'

.objects(x=x, y=y)
```

Arguments

objects

Any objects by name and value.

Returns

Self [OperationalizationDefinition](#) for fluent API chaining.

outputs

Python

```
outputs(**outputs)
```

Defines a service's optional supported outputs by name and type.

Example:

```
.outputs(a=float, b=int, c=str, d=bool, e='pandas.DataFrame')
```

Arguments

outputs

The outputs by name and type.

Returns

Self [OperationalizationDefinition](#) for fluent API chaining.

redeploy

Python

```
redeploy(force=False)
```

Bundle up the definition properties and update the service.

Returns

A new instance of [Service](#) representing the service *deployed*.

version

Python

```
version(version)
```

Set the service's optional version.

Arguments

version

The version of the service.

Returns

Self [OperationalizationDefinition](#) for fluent API.

Class RealtimeDefinition

Article • 02/28/2023

RealtimeDefinition

```
azureml.deploy.operationalization.RealtimeDefinition(name, op)
```

Bases: [azureml.deploy.operationalization.OperationalizationDefinition](#)

Real-time class defining a *real-time* service's properties for publishing.

Python

```
alias(alias)
```

Set the optional service function name alias to use in order to consume the service.

Example:

```
service = client.service('score-service').alias('score').deploy()  
  
# `score()` is the function that will call the `score-service`  
result = service.score()
```

Arguments

alias

The service function name alias to use in order to consume the service.

Returns

Self [OperationalizationDefinition](#) for fluent API.

deploy

```
Python
```

```
deploy()
```

Bundle up the definition properties and publish the service.

Returns

A new instance of [Service](#) representing the service *deployed*.

description

```
Python
```

```
description(description)
```

Set the service's optional description.

Arguments

description

The description of the service.

Returns

Self [OperationalizationDefinition](#) for fluent API.

redeploy

```
Python
```

```
redeploy(force=False)
```

Bundle up the definition properties and update the service.

Returns

A new instance of [Service](#) representing the service *deployed*.

serialized_model

Python

```
serialized_model(model)
```

Serialized model.

Arguments

model

The required serialized model used for this real-time service.

Returns

Self [OperationalizationDefinition](#) for fluent API chaining.

version

Python

```
version(version)
```

Set the service's optional version.

Arguments

version

The version of the service.

Returns

Self [OperationalizationDefinition](#) for fluent API.

Class Service

Article • 02/28/2023

Service

```
azureml.deploy.server.service.Service(service, http_client)
```

Dynamic object for service consumption and batching based on service metadata attributes.

batch

```
batch(records, parallel_count=10)
```

Register a set of input records for batch execution on this service.

Arguments

records

The *data.frame* or *list* of input records to execute.

parallel_count

Number of threads used to process entries in the batch. Default value is 10. Please make sure not to use too high of a number because it might negatively impact performance.

Returns

The [Batch](#) instance to control this service's batching lifecycle.

capabilities

Python

```
capabilities()
```

Provides the following information describing the holdings of this service:

- *api* - The API REST endpoint.
- *name* - The service name.
- *version* - The service version.
- *published_by* - The service publishing author.
- *runtime* - The service runtime context *R|Python*.
- *description* - The service description.
- *creation_time* - The service publish timestamp.
- *snapshot_id* - The snapshot identifier this service is bound with.
- *inputs* - The input schema name/type definition.
- *outputs* - The output schema name/type definition.
- *inputs_encoded* - The input schema name/type encoded to python.
- *outputs_encoded* - The output schema name/type encoded to python.
- *artifacts* - The supported generated files.
- *operation_id* - The function `alias`.
- *swagger* - The API REST endpoint to this service's *swagger.json* document.

Returns

A `dict` of key/values describing the service.

get_batch

Python

```
get_batch(execution_id)
```

Retrieves the service batch based on an execution identifier.

Arguments

execution_id

The identifier of the batch execution.

Returns

The [Batch](#) instance to control this service's batching lifecycle.

list_executions

```
Python
```

```
list_batch_executions()
```

Gets all batch execution identifiers currently queued for this service.

Returns

A `list` of execution identifiers.

swagger

```
Python
```

```
swagger()
```

Retrieves the *swagger.json* for this service (see <http://swagger.io/> [↗](#)).

Returns

The swagger document for this service as a json `str`.

Class ServiceResponse

Article • 02/28/2023

ServiceResponse

```
azureml.deploy.server.service.ServiceResponse(api, response, output_schema)
```

Represents the response from a service invocation. The response will contain any outputs and file artifacts produced in addition to any console output or errors messages.

api

```
Python
```

```
api
```

Gets the api endpoint.

artifact

```
Python
```

```
artifact(artifact_name, decode=True, encoding=None)
```

A convenience function to look up a file artifact by name and optionally base64 decode it.

Arguments

artifact_name

The name of the file artifact.

decode

Whether to decode the Base64 encoded artifact string. The default is `True`.

encoding

The encoding scheme to be used. The default is to apply no encoding. For a list of all encoding schemes please visit *Standard Encodings*:

<https://docs.python.org/3/library/codecs.html#standard-encodings> ↗

Returns

The file artifact as a Base64 encoded string if `decode=False` otherwise the decoded string.

artifacts

```
Python
```

```
artifacts
```

Returns a `list` of non-decoded response file artifacts if present.

console_output

```
Python
```

```
console_output
```

Gets the console output if present.

error

```
Python
```

```
error
```

Gets the error if present.

output

```
Python
```

```
output(output)
```

A convenience function to look up an output value by name.

Arguments

output

The name of the output.

Returns

The service output's value.

outputs

```
Python
```

```
outputs
```

Gets the response outputs if present.

raw_outputs

```
Python
```

```
raw_outputs
```

Gets the raw response outputs if present.

Class Batch

Article • 02/28/2023

Batch

```
azureml.deploy.server.service.Batch(service, records=[], parallel_count=10,
    execution_id=None)
```

Manager of a service's batch execution lifecycle.

api

```
Python
```

```
api
```

Gets the api endpoint.

execution_id

```
Python
```

```
execution_id
```

Gets this batch's execution identifier if currently started, otherwise `None`.

parallel_count

```
Python
```

```
parallel_count
```

Gets this batch's parallel count of threads.

```
records
```

Gets the batch input records.

```
results(show_partial_results=True)
```

Poll for batch results.

Arguments

show_partial_results

To get partial execution results or not. The default is to include partial results.

Returns

An instance of [BatchResponse](#).

start

```
Python
```

```
start()
```

Starts a batch execution for this service.

Returns

An instance of itself [Batch](#).

artifacts

```
Python
```

```
artifact(index, file_name)
```

Get the file artifact for this service batch execution *index*.

Arguments

index

Batch execution index.

file_name

Artifact filename

Returns

A single file artifact.

cancel

```
Python
```

```
cancel()
```

Cancel this batch execution.

download

```
download(index, file_name=None, destination=cwd())
```

Download the file artifact to file-system in the *destination*.

Arguments

index

Batch execution index.

file_name

The file artifact name.

destination

Download location.

Returns

A *list* of downloaded file-paths.

list_artifacts

```
Python
```

```
list_artifacts(index)
```

List the file artifact names belonging to this service batch execution *index*.

Arguments

index

Batch execution index.

Returns

A *list* of file artifact names.

Gets this batch's parallel count of threads.

records

```
Python
```

```
records
```

Gets the batch input records.

results

```
Python
```

```
results(show_partial_results=True)
```

Poll batch results.

Arguments

`show_partial_results`

To get partial execution results or not.

Returns

An execution Self *BatchResponse*.

Class BatchResponse

Article • 02/28/2023

BatchResponse

```
azureml.deploy.server.service.BatchResponse(api, execution_id, response,
      output_schema)
```

Represents a service's entire batch execution response at a particular state in time. Using this, a batch execution index can be supplied to the `execution(index)` function in order to retrieve the service's [ServiceResponse](#).

api

```
Python
```

```
api
```

Gets the api endpoint.

completed_item_count

```
Python
```

```
completed_item_count
```

Returns the number of completed batch results processed thus far.

execution

```
Python
```

```
execution(index)
```

Extracts the service execution results within the batch at this execution *index*.

Arguments

index

The batch execution index.

Returns

The execution results [ServiceResponse](#).

execution_id

```
Python
```

```
execution_id
```

Returns this batch's execution identifier if a batch has been started, otherwise `None`.

total_item_count

```
Python
```

```
total_item_count
```



Returns the total number of batch results processed in any state.

microsoftml (Python package in SQL Server Machine Learning Services)

Article • 02/28/2023

Applies to:  SQL Server 2017 (14.x) and later

microsoftml is a Python package from Microsoft that provides high-performance machine learning algorithms. It includes functions for training and transformations, scoring, text and image analysis, and feature extraction for deriving values from existing data. The package is included in [SQL Server Machine Learning Services](#) and supports high performance on big data, using multicore processing, and fast data streaming.

Package details	Information
Current version:	9.4
Built on:	Anaconda 4.2  distribution of Python 3.7.1 
Package distribution:	SQL Server Machine Learning Services version 2017 or 2019.

How to use microsoftml

The **microsoftml** module is installed as part of SQL Server Machine Learning Services when you add Python to your installation. You get the full collection of proprietary packages plus a Python distribution with its modules and interpreters. You can use any Python IDE to write Python script calling functions in **microsoftml**, but the script must run on a computer having SQL Server Machine Learning Services with Python.

Microsoftml and **revoscalepy** are tightly coupled; data sources used in **microsoftml** are defined as **revoscalepy** objects. Compute context limitations in **revoscalepy** transfer to **microsoftml**. Namely, all functionality is available for local operations, but switching to a remote compute context requires [RxSpark](#) or [RxInSQLServer](#).

Versions and platforms

The **microsoftml** module is available only when you install one of the following Microsoft products or downloads:

- [SQL Server Machine Learning Services](#)
- [Python client libraries for a data science client](#)



ⓘ Note

Full product release versions are Windows-only in SQL Server 2017. Both Windows and Linux are supported for **microsoftml** in **SQL Server 2019**.

Package dependencies

Algorithms in **microsoftml** depend on [revoscalepy](#) for:

- Data source objects - Data consumed by **microsoftml** functions are created using **revoscalepy** functions.
- Remote computing (shifting function execution to a remote SQL Server instance) - The **revoscalepy** package provides functions for creating and activating a remote compute context for SQL server.

In most cases, you will load the packages together whenever you are using **microsoftml**.

Functions by category

This section lists the functions by category to give you an idea of how each one is used. You can also use the table of contents to find functions in alphabetical order.

1-Training functions

Function	Description
microsoftml.rx_ensemble	Train an ensemble of models.
microsoftml.rx_fast_forest	Random Forest.
microsoftml.rx_fast_linear	Linear Model. with Stochastic Dual Coordinate Ascent.
microsoftml.rx_fast_trees	Boosted Trees.
microsoftml.rx_logistic_regression	Logistic Regression.
microsoftml.rx_neural_network	Neural Network.
microsoftml.rx_oneclass_svm	Anomaly Detection.

2-Transform functions

Categorical variable handling

Function	Description
microsoftml.categorical	Converts a text column into categories.
microsoftml.categorical_hash	Hashes and converts a text column into categories.

Schema manipulation

Function	Description
microsoftml.concat	Concatenates multiple columns into a single vector.
microsoftml.drop_columns	Drops columns from a dataset.
microsoftml.select_columns	Retains columns of a dataset.

Variable selection

Function	Description
microsoftml.count_select	Feature selection based on counts.
microsoftml.mutualinformation_select	Feature selection based on mutual information.

Text analytics

Function	Description
microsoftml.featurize_text	Converts text columns into numerical features.
microsoftml.get_sentiment	Sentiment analysis.

Image analytics

Function	Description
microsoftml.load_image	Loads an image.
microsoftml.resize_image	Resizes an Image.
microsoftml.extract_pixels	Extracts pixels from an image.

Function	Description
microsoftml.featurize_image	Converts an image into features.

Featurization functions

Function	Description
microsoftml.rx_featurize	Data transformation for data sources

Scoring functions

Function	Description
microsoftml.rx_predict	Scores using a Microsoft machine learning model

How to call microsoftml

Functions in **microsoftml** are callable in Python code encapsulated in stored procedures. Most developers build **microsoftml** solutions locally, and then migrate finished Python code to stored procedures as a deployment exercise.

The **microsoftml** package for Python is installed by default, but unlike **revoscalepy**, it is not loaded by default when you start a Python session using the Python executables installed with SQL Server.

As a first step, import the **microsoftml** package, and import **revoscalepy** if you need to use remote compute contexts or related connectivity or data source objects. Then, reference the individual functions you need.

Python

```
from microsoftml.modules.logistic_regression.rx_logistic_regression import rx_logistic_regression
from revoscalepy.functions.RxSummary import rx_summary
from revoscalepy.etl.RxImport import rx_import_datasource
```

See also

- [Python tutorials](#)
- [Manage Python packages](#)

microsoftml.adadelta_optimizer: Adaptive learning rate method

Article • 03/03/2023

Usage

```
microsoftml.adadelta_optimizer(decay: numbers.Real = 0.95,  
                               cond: numbers.Real = 1e-06)
```

Description

Adaptive learning rate method.

Arguments

decay

Decay rate (settings).

cond

Condition constant (settings).

See also

[sgd_optimizer](#)

microsoftml.avx_math: Acceleration with AVX instructions

Article • 03/03/2023

Usage

```
microsoftml.avx_math()
```

Description

Implementation accelerated with AVX instructions.

See also

[clr_math](#), [gpu_math](#), [mkl_math](#), [sse_math](#)

microsoftml.categorical: Converts a text column into categories

Article • 03/03/2023

Usage

```
microsoftml.categorical(cols: [str, dict, list], output_kind: ['Bag', 'Ind', 'Key', 'Bin'] = 'Ind', max_num_terms: int = 1000000, terms: int = None, sort: ['Occurrence', 'Value'] = 'Occurrence', text_key_values: bool = False, **kargs)
```

Description

Categorical transform that can be performed on data before training a model.

Details

The `categorical` transform passes through a data set, operating on text columns, to build a dictionary of categories. For each row, the entire text string appearing in the input column is defined as a category. The output of the categorical transform is an indicator vector. Each slot in this vector corresponds to a category in the dictionary, so its length is the size of the built dictionary. The categorical transform can be applied to one or more columns, in which case it builds a separate dictionary for each column that it is applied to.

`categorical` is not currently supported to handle factor data.

Arguments

`cols`

A character string or list of variable names to transform. If `dict`, the keys represent the names of new variables to be created.

`output_kind`

A character string that specifies the kind of output kind.

- `"Bag"`: Outputs a multi-set vector. If the input column is a vector of categories, the output contains one vector, where the value in each slot is the number of occurrences of the category in the input vector. If the input column contains a single category, the indicator vector and the bag vector are equivalent
- `"Ind"`: Outputs an indicator vector. The input column is a vector of categories, and the output contains one indicator vector per slot in the input column.
- `"Key"`: Outputs an index. The output is an integer ID (between 1 and the number of categories in the dictionary) of the category.
- `"Bin"`: Outputs a vector which is the binary representation of the category.

The default value is `"Ind"`.

max_num_terms

An integer that specifies the maximum number of categories to include in the dictionary. The default value is 1000000.

terms

Optional character vector of terms or categories.

sort

A character string that specifies the sorting criteria.

- `"Occurrence"`: Sort categories by occurrences. Most frequent is first.
- `"Value"`: Sort categories by values.

text_key_values

Whether key value metadata should be text, regardless of the actual input type.

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

[categorical_hash](#)

Example

```
...
Example on rx_logistic_regression and categorical.
...

import numpy
import pandas
from microsoftml import rx_logistic_regression, categorical, rx_predict

train_reviews = pandas.DataFrame(data=dict(
    review=[
        "This is great", "I hate it", "Love it", "Do not like it", "Really
like it",
        "I hate it", "I like it a lot", "I kind of hate it", "I do like it",
        "I really hate it", "It is very good", "I hate it a bunch", "I love
it a bunch",
        "I hate it", "I like it very much", "I hate it very much.",
        "I really do love it", "I really do hate it", "Love it!", "Hate
it!",
        "I love it", "I hate it", "I love it", "I hate it", "I love it"],
    like=[True, False, True, False, True, False, True, False, True, False,
        True, False, True, False, True, False, True, False, True, False,
        True,
        False, True, False, True]))

test_reviews = pandas.DataFrame(data=dict(
    review=[
        "This is great", "I hate it", "Love it", "Really like it", "I hate
it",
        "I like it a lot", "I love it", "I do like it", "I really hate it",
"I love it"]]))

# Use a categorical transform: the entire string is treated as a category
out_model = rx_logistic_regression("like ~ reviewCat",
    data=train_reviews,
    ml_transforms=[categorical(cols=dict(reviewCat="review"))])

# Note that 'I hate it' and 'I love it' (the only strings appearing more
than once)
# have non-zero weights.
```



```

print(out_model.coef_)

# Use the model to score.
source_out_df = rx_predict(out_model, data=test_reviews,
extra_vars_to_write=["review"])
print(source_out_df.head())

```

Output:

```

Beginning processing data.
Rows Read: 25, Read Time: 0, Transform Time: 0
Beginning processing data.
Not adding a normalizer.
Beginning processing data.
Rows Read: 25, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 25, Read Time: 0, Transform Time: 0
Beginning processing data.
LBFGS multi-threading will attempt to load dataset into memory. In case of
out-of-memory issues, turn off multi-threading by setting trainThreads to 1.
Warning: Too few instances to use 4 threads, decreasing to 1 thread(s)
Beginning optimization
num vars: 20
improvement criterion: Mean Improvement
L1 regularization selected 3 of 20 weights.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:01.6550695
Elapsed time: 00:00:00.2259981
OrderedDict([('Bias', 0.21317288279533386), ('I hate it',
-0.7937591671943665), ('I love it', 0.19668534398078918)])
Beginning processing data.
Rows Read: 10, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.1385248
Finished writing 10 rows.
Writing completed.

```

	review	PredictedLabel	Score	Probability
0	This is great	True	0.213173	0.553092
1	I hate it	False	-0.580586	0.358798
2	Love it	True	0.213173	0.553092
3	Really like it	True	0.213173	0.553092
4	I hate it	False	-0.580586	0.358798

microsoftml.categorical_hash: Hashes and converts a text column into categories

Article • 03/03/2023

Usage

```
microsoftml.categorical_hash(cols: [str, dict, list],
    hash_bits: int = 16, seed: int = 314489979,
    ordered: bool = True, invert_hash: int = 0,
    output_kind: ['Bag', 'Ind', 'Key', 'Bin'] = 'Bag', **kwargs)
```

Description

Categorical hash transform that can be performed on data before training a model.

Details

`categorical_hash` converts a categorical value into an indicator array by hashing the value and using the hash as an index in the bag. If the input column is a vector, a single indicator bag is returned for it. `categorical_hash` does not currently support handling factor data.

Arguments

cols

A character string or list of variable names to transform. If `dict`, the keys represent the names of new variables to be created.

hash_bits

An integer specifying the number of bits to hash into. Must be between 1 and 30, inclusive. The default value is 16.

seed

An integer specifying the hashing seed. The default value is 314489979.

ordered

`True` to include the position of each term in the hash. Otherwise, `False`. The default value is `True`.

invert_hash

An integer specifying the limit on the number of keys that can be used to generate the slot name. `0` means no invert hashing; `-1` means no limit. While a zero value gives better performance, a non-zero value is needed to get meaningful coefficient names. The default value is `0`.

output_kind

A character string that specifies the kind of output kind.

- `"Bag"`: Outputs a multi-set vector. If the input column is a vector of categories, the output contains one vector, where the value in each slot is the number of occurrences of the category in the input vector. If the input column contains a single category, the indicator vector and the bag vector are equivalent
- `"Ind"`: Outputs an indicator vector. The input column is a vector of categories, and the output contains one indicator vector per slot in the input column.
- `"Key"`: Outputs an index. The output is an integer ID (between 1 and the number of categories in the dictionary) of the category.
- `"Bin"`: Outputs a vector which is the binary representation of the category.

The default value is `"Bag"`.

kargs

Additional arguments sent to the compute engine.

Returns

an object defining the transform.

See also

[categorical](#)

Example

```
...
Example on rx_logistic_regression and categorical_hash.
...

import numpy
import pandas
from microsoftml import rx_logistic_regression, categorical_hash, rx_predict
from microsoftml.datasets.datasets import get_dataset

movie_reviews = get_dataset("movie_reviews")

train_reviews = pandas.DataFrame(data=dict(
    review=[
        "This is great", "I hate it", "Love it", "Do not like it", "Really
like it",
        "I hate it", "I like it a lot", "I kind of hate it", "I do like it",
        "I really hate it", "It is very good", "I hate it a bunch", "I love
it a bunch",
        "I hate it", "I like it very much", "I hate it very much.",
        "I really do love it", "I really do hate it", "Love it!", "Hate
it!",
        "I love it", "I hate it", "I love it", "I hate it", "I love it"],
    like=[True, False, True, False, True, False, True, False, True, False,
        True, False, True, False, True, False, True, False, True, False,
        True,
        False, True, False, True]))

test_reviews = pandas.DataFrame(data=dict(
    review=[
        "This is great", "I hate it", "Love it", "Really like it", "I hate
it",
        "I like it a lot", "I love it", "I do like it", "I really hate it",
"I love it"]]))

# Use a categorical hash transform.
out_model = rx_logistic_regression("like ~ reviewCat",
    data=train_reviews,
    ml_transforms=
[categorical_hash(cols=dict(reviewCat="review"))])
```

```
# Weights are similar to categorical.
print(out_model.coef_)

# Use the model to score.
source_out_df = rx_predict(out_model, data=test_reviews,
extra_vars_to_write=["review"])
print(source_out_df.head())
```

Output:

```
Not adding a normalizer.
Beginning processing data.
Rows Read: 25, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 25, Read Time: 0, Transform Time: 0
Beginning processing data.
LBFGS multi-threading will attempt to load dataset into memory. In case of
out-of-memory issues, turn off multi-threading by setting trainThreads to 1.
Warning: Too few instances to use 4 threads, decreasing to 1 thread(s)
Beginning optimization
num vars: 65537
improvement criterion: Mean Improvement
L1 regularization selected 3 of 65537 weights.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.1209392
Elapsed time: 00:00:00.0190134
OrderedDict([('Bias', 0.2132447361946106), ('f1783', -0.7939924597740173),
('f38537', 0.1968022584915161)])
Beginning processing data.
Rows Read: 10, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0284223
Finished writing 10 rows.
Writing completed.
```

	review	PredictedLabel	Score	Probability
0	This is great	True	0.213245	0.553110
1	I hate it	False	-0.580748	0.358761
2	Love it	True	0.213245	0.553110
3	Really like it	True	0.213245	0.553110
4	I hate it	False	-0.580748	0.358761

microsoftml.clr_math: Acceleration with .NET math

Article • 03/03/2023

Usage

```
microsoftml.clr_math()
```

Description

Default .NET math.

See also

[avx_math](#), [gpu_math](#), [mkl_math](#), [sse_math](#)

microsoftml.concat: Concatenates multiple columns into a single vector

Article • 03/03/2023

Usage

```
microsoftml.concat(cols: [dict, list], **kargs)
```

Description

Combines several columns into a single vector-valued column.

Details

`concat` creates a single vector-valued column from multiple columns. It can be performed on data before training a model. The concatenation can significantly speed up the processing of data when the number of columns is as large as hundreds to thousands.

Arguments

cols

A character dict or list of variable names to transform. If `dict`, the keys represent the names of new variables to be created. Note that all the input variables must be of the same type. It is possible to produce multiple output columns with the concatenation transform. In this case, you need to use a list of vectors to define a one-to-one mapping between input and output variables. For example, to concatenate columns `InNameA` and `InNameB` into column `OutName1` and also columns `InNameC` and `InNameD` into column `OutName2`, use the dict: `dict(OutName1 = [InNameA, InNameB], outName2 = [InNameC, InNameD])`

kargs

Additional arguments sent to the compute engine.

Returns

An object defining the concatenation transform.

See also

[drop_columns](#), [select_columns](#).

Example

```
...
Example on logistic regression and concat.
...
import numpy
import pandas
import sklearn
from microsoftml import rx_logistic_regression, concat, rx_predict
from microsoftml.datasets.datasets import get_dataset

iris = get_dataset("iris")

if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

# We use iris dataset.
irisdf = iris.as_df()

# The training features.
features = ["Sepal_Length", "Sepal_Width", "Petal_Length", "Petal_Width"]

# The label.
label = "Label"

# microsoftml needs a single dataframe with features and label.
cols = features + [label]

# We split into train/test. y_train, y_test are not used.
data_train, data_test, y_train, y_test = train_test_split(irisdf[cols],
irisdf[label])

# We train a logistic regression.
# A concat transform is added to group features in a single vector column.
multi_logit_out = rx_logistic_regression(
```



```

        formula="Label ~ Features",
        method="multiClass",
        data=data_train,
        ml_transforms=[concat(cols={'Features': features})])

# We show the coefficients.
print(multi_logit_out.coef_)

# We predict.
prediction = rx_predict(multi_logit_out, data=data_test)

print(prediction.head())

```

Output:

```

Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 112, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 112, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 112, Read Time: 0.001, Transform Time: 0
Beginning processing data.
LBFGS multi-threading will attempt to load dataset into memory. In case of
out-of-memory issues, turn off multi-threading by setting trainThreads to 1.
Beginning optimization
num vars: 15
improvement criterion: Mean Improvement
L1 regularization selected 9 of 15 weights.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.2348578
Elapsed time: 00:00:00.0197433
OrderedDict([('0+(Bias)', 1.943994402885437), ('1+(Bias)',
0.6346845030784607), ('2+(Bias)', -2.57867693901062), ('0+Petal_Width',
-2.7277402877807617), ('0+Petal_Length', -2.5394322872161865),
('0+Sepal_Width', 0.4810805320739746), ('1+Sepal_Width',
-0.5790582299232483), ('2+Petal_Width', 2.547518491744995),
('2+Petal_Length', 1.6753791570663452)])
Beginning processing data.
Rows Read: 38, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0662932
Finished writing 38 rows.
Writing completed.
    Score.0  Score.1  Score.2
0  0.320061  0.504115  0.175825
1  0.761624  0.216213  0.022163
2  0.754765  0.215548  0.029687

```

3	0.182810	0.517855	0.299335
4	0.018770	0.290014	0.691216

microsoftml.count_select: Feature selection based on counts

Article • 03/03/2023

Usage

```
microsoftml.count_select(cols: [list, str], count: int = 1, **kargs)
```

Description

Selects the features for which the count of non-default values is greater than or equal to a threshold.

Details

When using the count mode in feature selection transform, a feature is selected if the number of examples have at least the specified count examples of non-default values in the feature. The count mode feature selection transform is very useful when applied together with a categorical hash transform (see also, `categorical_hash`). The count feature selection can remove those features generated by hash transform that have no data in the examples.

Arguments

cols

Specifies character string or list of the names of the variables to select.

count

The threshold for count based feature selection. A feature is selected if and only if at least `count` examples have non-default value in the feature. The default value is 1.

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

[mutualinformation_select](#)

microsoftml.custom: Removes custom stopwords

Article • 02/28/2023

Usage

```
microsoftml.custom(stopword: list = None)
```

Description

Remover with list of stopwords specified by the user.

Arguments

stopword

List of stopwords (settings).

microsoftml.drop_columns: Drops columns from a dataset

Article • 03/03/2023

Usage

```
microsoftml.drop_columns(cols: [list, str], **kargs)
```

Description

Specified columns to drop from the dataset.

Arguments

cols

A character string or list of the names of the variables to drop.

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

[concat](#), [select_columns](#).

microsoftml.extract_pixels: Extracts pixels from an image

Article • 03/03/2023

Usage

```
microsoftml.extract_pixels(cols: [str, dict, list],
    use_alpha: bool = False, use_red: bool = True,
    use_green: bool = True, use_blue: bool = True,
    interleave_argb: bool = False, convert: bool = True,
    offset: float = None, scale: float = None, **kargs)
```

Description

Extracts the pixel values from an image.

Details

`extract_pixels` extracts the pixel values from an image. The input variables are images of the same size, typically the output of a `resizeImage` transform. The output is pixel data in vector form that are typically used as features for a learner.

Arguments

cols

A character string or list of variable names to transform. If `dict`, the keys represent the names of new variables to be created.

use_alpha

Specifies whether to use alpha channel. The default value is `False`.

use_red

Specifies whether to use red channel. The default value is `True`.

use_green

Specifies whether to use green channel. The default value is `True`.

use_blue

Specifies whether to use blue channel. The default value is `True`.

interleave_argb

Whether to separate each channel or interleave in ARGB order. This might be important, for example, if you are training a convolutional neural network, since this would affect the shape of the kernel, stride etc.

convert

Whether to convert to floating point. The default value is `False`.

offset

Specifies the offset (pre-scale). This requires `convert = True`. The default value is *None*.

scale

Specifies the scale factor. This requires `convert = True`. The default value is *None*.

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

[load_image](#), [resize_image](#), [featurize_image](#).

Example

```
...
Example with images.
...
import numpy
import pandas
from microsoftml import rx_neural_network, rx_predict, rx_fast_linear
from microsoftml import load_image, resize_image, extract_pixels
from microsoftml.datasets.image import get_RevolutionAnalyticslogo

train = pandas.DataFrame(data=dict(Path=[get_RevolutionAnalyticslogo()],
Label=[True]))

# Loads the images from variable Path, resizes the images to 1x1 pixels
# and trains a neural net.
model1 = rx_neural_network("Label ~ Features", data=train,
    ml_transforms=[
        load_image(cols=dict(Features="Path")),
        resize_image(cols="Features", width=1, height=1,
resizing="Aniso"),
        extract_pixels(cols="Features")],
    ml_transform_vars=["Path"],
    num_hidden_nodes=1, num_iterations=1)

# Featurizes the images from variable Path using the default model, and
# trains a linear model on the result.
# If dnnModel == "AlexNet", the image has to be resized to 227x227.
model2 = rx_fast_linear("Label ~ Features ", data=train,
    ml_transforms=[
        load_image(cols=dict(Features="Path")),
        resize_image(cols="Features", width=224, height=224),
        extract_pixels(cols="Features")],
    ml_transform_vars=["Path"], max_iterations=1)

# We predict even if it does not make too much sense on this single image.
print("\nrx_neural_network")
prediction1 = rx_predict(model1, data=train)
print(prediction1)

print("\nrx_fast_linear")
prediction2 = rx_predict(model2, data=train)
print(prediction2)
```

Output:

```
Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
```

Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Using: AVX Math

***** Net definition *****

```
input Data [3];
hidden H [1] sigmoid { // Depth 1
  from Data all;
}
output Result [1] sigmoid { // Depth 0
  from H all;
}
```

***** End net definition *****

Input count: 3
Output count: 1
Output Function: Sigmoid
Loss Function: LogLoss
PreTrainer: NoPreTrainer

Starting training...
Learning rate: 0.001000
Momentum: 0.000000
InitWtsDiameter: 0.100000

Initializing 1 Hidden Layers, 6 Weights...
Estimated Pre-training MeanError = 0.707823
Iter:1/1, MeanErr=0.707823(0.00%), 0.00M WeightUpdates/sec
Done!
Estimated Post-training MeanError = 0.707499

Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.2716496
Elapsed time: 00:00:00.0396484
Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Using 2 threads to train.
Automatically choosing a check frequency of 2.
Auto-tuning parameters: L2 = 5.
Auto-tuning parameters: L1Threshold (L1/L2) = 1.

Using model from last iteration.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:01.0508885
Elapsed time: 00:00:00.0133784

rx_neural_network
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.1339430
Finished writing 1 rows.
Writing completed.

	PredictedLabel	Score	Probability
0	False	-0.028504	0.492875

rx_fast_linear
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.4977487
Finished writing 1 rows.
Writing completed.

	PredictedLabel	Score	Probability
0	False	0.0	0.5

microsoftml.featurize_image: Converts an image into features

Article • 03/03/2023

Usage

```
microsoftml.featurize_image(cols: [dict, str], dnn_model: ['Resnet18',  
  'Resnet50', 'Resnet101', 'Alexnet'] = 'Resnet18', **kwargs)
```

Description

Featurizes an image using a pre-trained deep neural network model.

Details

`featurize_image` featurizes an image using the specified pre-trained deep neural network model. The input variables to this transform must be extracted pixel values.

Arguments

cols

Input variable containing extracted pixel values. If `dict`, the keys represent the names of new variables to be created.

dnn_model

The pre-trained deep neural network. The possible options are:

- `"Resnet18"`
- `"Resnet50"`
- `"Resnet101"`

- "Alexnet"

The default value is "Resnet18". See [Deep Residual Learning for Image Recognition](#) for details about ResNet.

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

[load_image](#), [resize_image](#), [extract_pixels](#).

Example

```
...
Example with images.
...
import numpy
import pandas
from microsoftml import rx_neural_network, rx_predict, rx_fast_linear
from microsoftml import load_image, resize_image, extract_pixels
from microsoftml.datasets.image import get_RevolutionAnalyticslogo

train = pandas.DataFrame(data=dict(Path=[get_RevolutionAnalyticslogo()],
Label=[True]))

# Loads the images from variable Path, resizes the images to 1x1 pixels
# and trains a neural net.
model1 = rx_neural_network("Label ~ Features", data=train,
    ml_transforms=[
        load_image(cols=dict(Features="Path")),
        resize_image(cols="Features", width=1, height=1,
resizing="Aniso"),
        extract_pixels(cols="Features")],
    ml_transform_vars=["Path"],
    num_hidden_nodes=1, num_iterations=1)

# Featurizes the images from variable Path using the default model, and
# trains a linear model on the result.
```

```

# If dnnModel == "AlexNet", the image has to be resized to 227x227.
model2 = rx_fast_linear("Label ~ Features ", data=train,
                        ml_transforms=[
                            load_image(cols=dict(Features="Path")),
                            resize_image(cols="Features", width=224, height=224),
                            extract_pixels(cols="Features")],
                        ml_transform_vars=["Path"], max_iterations=1)

# We predict even if it does not make too much sense on this single image.
print("\nrx_neural_network")
prediction1 = rx_predict(model1, data=train)
print(prediction1)

print("\nrx_fast_linear")
prediction2 = rx_predict(model2, data=train)
print(prediction2)

```

Output:

```

Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Using: AVX Math

***** Net definition *****
  input Data [3];
  hidden H [1] sigmoid { // Depth 1
    from Data all;
  }
  output Result [1] sigmoid { // Depth 0
    from H all;
  }
***** End net definition *****
Input count: 3
Output count: 1
Output Function: Sigmoid
Loss Function: LogLoss
PreTrainer: NoPreTrainer



---


Starting training...
Learning rate: 0.001000
Momentum: 0.000000
InitWtsDiameter: 0.100000

```

Initializing 1 Hidden Layers, 6 Weights...
Estimated Pre-training MeanError = 0.707823
Iter:1/1, MeanErr=0.707823(0.00%), 0.01M WeightUpdates/sec
Done!
Estimated Post-training MeanError = 0.707499

Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.0751759
Elapsed time: 00:00:00.0080433
Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Using 2 threads to train.
Automatically choosing a check frequency of 2.
Auto-tuning parameters: L2 = 5.
Auto-tuning parameters: L1Threshold (L1/L2) = 1.
Using model from last iteration.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:01.0104773
Elapsed time: 00:00:00.0106935

rx_neural_network
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0420328
Finished writing 1 rows.
Writing completed.

	PredictedLabel	Score	Probability
0	False	-0.028504	0.492875

rx_fast_linear
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.4449623
Finished writing 1 rows.
Writing completed.

	PredictedLabel	Score	Probability
0	False	0.0	0.5

microsoftml.featurize_text: Converts text columns into numerical features

Article • 03/03/2023

Usage

```
microsoftml.featurize_text(cols: [str, dict, list], language: ['AutoDetect',
    'English', 'French', 'German', 'Dutch', 'Italian', 'Spanish',
    'Japanese'] = 'English', stopwords_removal=None, case: ['Lower',
    'Upper', 'None'] = 'Lower', keep_diacritics: bool = False,
    keep_punctuations: bool = True, keep_numbers: bool = True,
    dictionary: dict = None, word_feature_extractor={'Name': 'NGram',
    'Settings': {'Weighting': 'Tf', 'MaxNumTerms': [1000000],
    'NgramLength': 1, 'AllLengths': True, 'SkipLength': 0}},
    char_feature_extractor=None, vector_normalizer: ['None', 'L1', 'L2',
    'LInf'] = 'L2', **kwargs)
```

Description

Text transforms that can be performed on data before training a model.

Details

The `featurize_text` transform produces a bag of counts of sequences of consecutive words, called n-grams, from a given corpus of text. There are two ways it can do this:

- build a dictionary of n-grams and use the ID in the dictionary as the index in the bag;
- hash each n-gram and use the hash value as the index in the bag.

The purpose of hashing is to convert variable-length text documents into equal-length numeric feature vectors, to support dimensionality reduction and to make the lookup of feature weights faster.

The text transform is applied to text input columns. It offers language detection, tokenization, stopwords removing, text normalization and feature generation. It supports the following languages by default: English, French, German, Dutch, Italian, Spanish and Japanese.

The n-grams are represented as count vectors, with vector slots corresponding either to n-grams (created using `n_gram`) or to their hashes (created using `n_gram_hash`).

Embedding ngrams in a vector space allows their contents to be compared in an efficient manner. The slot values in the vector can be weighted by the following factors:

- *term frequency* - The number of occurrences of the slot in the text
- *inverse document frequency* - A ratio (the logarithm of inverse relative slot frequency) that measures the information a slot provides by determining how common or rare it is across the entire text.
- *term frequency-inverse document frequency* - the product term frequency and the inverse document frequency.

Arguments

cols

A character string or list of variable names to transform. If `dict`, the keys represent the names of new variables to be created.

language

Specifies the language used in the data set. The following values are supported:

- `"AutoDetect"`: for automatic language detection.
- `"English"`
- `"French"`
- `"German"`
- `"Dutch"`
- `"Italian"`
- `"Spanish"`
- `"Japanese"`

stopwords_remover

Specifies the stopwords remover to use. There are three options supported:

- *None*: No stopwords remover is used.
- `predefined`: A precompiled language-specific list of stop words is used that includes the most common words from Microsoft Office.
- `custom`: A user-defined list of stopwords. It accepts the following option: `stopword`.

The default value is *None*.

case

Text casing using the rules of the invariant culture. Takes the following values:

- `"Lower"`
- `"Upper"`
- `"None"`

The default value is `"Lower"`.

keep_diacritics

`False` to remove diacritical marks; `True` to retain diacritical marks. The default value is `False`.

keep_punctuations

`False` to remove punctuation; `True` to retain punctuation. The default value is `True`.

keep_numbers

`False` to remove numbers; `True` to retain numbers. The default value is `True`.

dictionary

A dictionary of allowlisted terms which accepts the following options:

- `term`: An optional character vector of terms or categories.
- `dropUnknowns`: Drop items.

- `sort`: Specifies how to order items when vectorized. Two orderings are supported:
 - `"occurrence"`: items appear in the order encountered.
 - `"value"`: items are sorted according to their default comparison. For example, text sorting will be case sensitive (e.g., 'A' then 'Z' then 'a').

The default value is *None*. Note that the stopwords list takes precedence over the dictionary allowlist as the stopwords are removed before the dictionary terms are allowlisted.

word_feature_extractor

Specifies the word feature extraction arguments. There are two different feature extraction mechanisms:

- `n_gram()`: Count-based feature extraction (equivalent to WordBag). It accepts the following options: `max_num_terms` and `weighting`.
- `n_gram_hash()`: Hashing-based feature extraction (equivalent to WordHashBag). It accepts the following options: `hash_bits`, `seed`, `ordered` and `invert_hash`.

The default value is `n_gram`.

char_feature_extractor

Specifies the char feature extraction arguments. There are two different feature extraction mechanisms:

- `n_gram()`: Count-based feature extraction (equivalent to WordBag). It accepts the following options: `max_num_terms` and `weighting`.
- `n_gram_hash()`: Hashing-based feature extraction (equivalent to WordHashBag). It accepts the following options: `hash_bits`, `seed`, `ordered` and `invert_hash`.

The default value is *None*.

vector_normalizer

Normalize vectors (rows) individually by rescaling them to unit norm. Takes one of the following values:

- `"None"`
- `"L2"`

- "L1"
- "LInf"

The default value is "L2".

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

[n_gram](#), [n_gram_hash](#), [n_gram](#), [n_gram_hash](#), [get_sentiment](#).

Example

```
...
Example with featurize_text and rx_logistic_regression.
...
import numpy
import pandas
from microsoftml import rx_logistic_regression, featurize_text, rx_predict
from microsoftml.entrypoints._stopwordsremover_predefined import predefined

train_reviews = pandas.DataFrame(data=dict(
    review=[
        "This is great", "I hate it", "Love it", "Do not like it", "Really
like it",
        "I hate it", "I like it a lot", "I kind of hate it", "I do like it",
        "I really hate it", "It is very good", "I hate it a bunch", "I love
it a bunch",
        "I hate it", "I like it very much", "I hate it very much.",
        "I really do love it", "I really do hate it", "Love it!", "Hate
it!",
        "I love it", "I hate it", "I love it", "I hate it", "I love it"],
    like=[True, False, True, False, True, False, True, False, True, False,
        True, False, True, False, True, False, True, False, True, False,
        True,
        False, True, False, True]))
```

```

test_reviews = pandas.DataFrame(data=dict(
    review=[
        "This is great", "I hate it", "Love it", "Really like it", "I hate
it",
        "I like it a lot", "I love it", "I do like it", "I really hate it",
"I love it"]]))

out_model = rx_logistic_regression("like ~ review_tran",
    data=train_reviews,
    ml_transforms=[
        featurize_text(cols=dict(review_tran="review"),
            stopwords_removal=predefined(),
            keep_punctuations=False)])

# Use the model to score.
score_df = rx_predict(out_model, data=test_reviews, extra_vars_to_write=
["review"])
print(score_df.head())

```

Output:

```

Beginning processing data.
Rows Read: 25, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 25, Read Time: 0, Transform Time: 0
Beginning processing data.
Not adding a normalizer.
Beginning processing data.
Rows Read: 25, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 25, Read Time: 0, Transform Time: 0
Beginning processing data.
LBFGS multi-threading will attempt to load dataset into memory. In case of
out-of-memory issues, turn off multi-threading by setting trainThreads to 1.
Warning: Too few instances to use 4 threads, decreasing to 1 thread(s)
Beginning optimization
num vars: 11
improvement criterion: Mean Improvement
L1 regularization selected 3 of 11 weights.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.3725934
Elapsed time: 00:00:00.0131199
Beginning processing data.
Rows Read: 10, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0635453
Finished writing 10 rows.
Writing completed.

```

	review	PredictedLabel	Score	Probability
0	This is great	True	0.443986	0.609208
1	I hate it	False	-0.668449	0.338844
2	Love it	True	0.994339	0.729944
3	Really like it	True	0.443986	0.609208
4	I hate it	False	-0.668449	0.338844

N-grams extractors

- *microsoftml.n_gram*: Converts text into features using n-grams
- *microsoftml.n_gram_hash*: Converts text into features using hashed n-grams

Stopwords removers

- *microsoftml.custom*: Removes custom stopwords
- *microsoftml.predefined*: Removes predefined stopwords

microsoftml.get_sentiment: Sentiment analysis

Article • 03/03/2023

Usage

```
microsoftml.get_sentiment(cols: [str, dict, list], **kargs)
```

Description

Scores natural language text and assesses the probability the sentiments are positive.

Details

The `get_sentiment` transform returns the probability that the sentiment of a natural text is positive. Currently supports only the English language.

Arguments

cols

A character string or list of variable names to transform. If `dict`, the names represent the names of new variables to be created.

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

Example

```
...
Example with get_sentiment and rx_logistic_regression.
...

import numpy
import pandas
from microsoftml import rx_logistic_regression, rx_featurize, rx_predict,
get_sentiment

# Create the data
customer_reviews = pandas.DataFrame(data=dict(review=[
    "I really did not like the taste of it",
    "It was surprisingly quite good!",
    "I will never ever ever go to that place again!!"]))

# Get the sentiment scores
sentiment_scores = rx_featurize(
    data=customer_reviews,
    ml_transforms=[get_sentiment(cols=dict(scores="review"))])

# Let's translate the score to something more meaningful
sentiment_scores["eval"] = sentiment_scores.scores.apply(
    lambda score: "AWESOMENESS" if score > 0.6 else "BLAH")
print(sentiment_scores)
```

Output:

```
Beginning processing data.
Rows Read: 3, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:02.4327924
Finished writing 3 rows.
Writing completed.
```

	review	scores	eval
0	I really did not like the taste of it	0.461790	BLAH
1	It was surprisingly quite good!	0.960192	AWESOMENESS
2	I will never ever ever go to that place again!!	0.310344	BLAH

microsoftml.gpu_math: Acceleration with NVidia CUDA

Article • 03/03/2023

Usage

```
microsoftml.gpu_math(gpu_id: numbers.Real = -1,  
                     cu_dnn: bool = False, cu_dnn_algo: str = 'ImplicitPrecompGemm')
```

Description

NVidia CUDA implementation.

Arguments

gpu_id

GPU device id (settings).

cu_dnn

Use cuDNN on GPU (settings).

cu_dnn_algo

cuDNN optimization options (settings).

See also

[avx_math](#), [clr_math](#), [mkl_math](#), [sse_math](#)

microsoftml.hinge_loss: Hinge loss function

Article • 03/03/2023

Usage

```
microsoftml.hinge_loss(margin: numbers.Real = 1.0)
```

Description

Hinge loss.

Arguments

margin

Margin value (settings).

See also

[log_loss](#), [smoothed_hinge_loss](#), [squared_loss](#)

microsoftml.load_image: Loads an image

Article • 03/03/2023

Usage

```
microsoftml.load_image(cols: [str, dict, list], **kargs)
```

Description

Loads image data.

Details

`load_image` loads images from paths.

Arguments

cols

A character string or list of variable names to transform. If `dict`, the keys represent the names of new variables to be created.

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

[resize_image](#), [extract_pixels](#), [featurize_image](#).

Example

```
'''
Example with images.
'''
import numpy
import pandas
from microsoftml import rx_neural_network, rx_predict, rx_fast_linear
from microsoftml import load_image, resize_image, extract_pixels
from microsoftml.datasets.image import get_RevolutionAnalyticslogo

train = pandas.DataFrame(data=dict(Path=[get_RevolutionAnalyticslogo()],
Label=[True]))

# Loads the images from variable Path, resizes the images to 1x1 pixels
# and trains a neural net.
model1 = rx_neural_network("Label ~ Features", data=train,
    ml_transforms=[
        load_image(cols=dict(Features="Path")),
        resize_image(cols="Features", width=1, height=1,
resizing="Aniso"),
        extract_pixels(cols="Features")],
    ml_transform_vars=["Path"],
    num_hidden_nodes=1, num_iterations=1)

# Featurizes the images from variable Path using the default model, and
# trains a linear model on the result.
# If dnnModel == "AlexNet", the image has to be resized to 227x227.
model2 = rx_fast_linear("Label ~ Features ", data=train,
    ml_transforms=[
        load_image(cols=dict(Features="Path")),
        resize_image(cols="Features", width=224, height=224),
        extract_pixels(cols="Features")],
    ml_transform_vars=["Path"], max_iterations=1)

# We predict even if it does not make too much sense on this single image.
print("\nrx_neural_network")
prediction1 = rx_predict(model1, data=train)
print(prediction1)

print("\nrx_fast_linear")
prediction2 = rx_predict(model2, data=train)
print(prediction2)
```

Output:

Automatically adding a MinMax normalization transform, use 'norm=Warn' or 'norm=No' to turn this behavior off.

Beginning processing data.

Rows Read: 1, Read Time: 0, Transform Time: 0

Beginning processing data.

Beginning processing data.

Rows Read: 1, Read Time: 0, Transform Time: 0

Beginning processing data.

Beginning processing data.

Rows Read: 1, Read Time: 0, Transform Time: 0

Beginning processing data.

Using: AVX Math

***** Net definition *****

```
input Data [3];
hidden H [1] sigmoid { // Depth 1
  from Data all;
}
output Result [1] sigmoid { // Depth 0
  from H all;
}
```

***** End net definition *****

Input count: 3

Output count: 1

Output Function: Sigmoid

Loss Function: LogLoss

PreTrainer: NoPreTrainer

Starting training...

Learning rate: 0.001000

Momentum: 0.000000

InitWtsDiameter: 0.100000

Initializing 1 Hidden Layers, 6 Weights...

Estimated Pre-training MeanError = 0.707823

Iter:1/1, MeanErr=0.707823(0.00%), 0.01M WeightUpdates/sec

Done!

Estimated Post-training MeanError = 0.707499

Not training a calibrator because it is not needed.

Elapsed time: 00:00:00.0891958

Elapsed time: 00:00:00.0095013

Automatically adding a MinMax normalization transform, use 'norm=Warn' or 'norm=No' to turn this behavior off.

Beginning processing data.

Rows Read: 1, Read Time: 0, Transform Time: 0

Beginning processing data.

Beginning processing data.

Rows Read: 1, Read Time: 0, Transform Time: 0

Beginning processing data.

Beginning processing data.

Rows Read: 1, Read Time: 0, Transform Time: 0

Beginning processing data.

Using 2 threads to train.

Automatically choosing a check frequency of 2.
Auto-tuning parameters: L2 = 5.
Auto-tuning parameters: L1Threshold (L1/L2) = 1.
Using model from last iteration.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:01.0541236
Elapsed time: 00:00:00.0113811

rx_neural_network

Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0401500
Finished writing 1 rows.
Writing completed.

	PredictedLabel	Score	Probability
0	False	-0.028504	0.492875

rx_fast_linear

Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.4957253
Finished writing 1 rows.
Writing completed.

	PredictedLabel	Score	Probability
0	False	0.0	0.5

microsoftml.log_loss: Log loss function

Article • 03/03/2023

Usage

```
microsoftml.log_loss()
```

Description

Log loss.

See also

[hinge_loss](#), [smoothed_hinge_loss](#), [squared_loss](#)

microsoftml.mkl_math: Acceleration with Intel MKL

Article • 03/03/2023

Usage

```
microsoftml.mkl_math(max_cache_size: numbers.Real = -1)
```

Description

Intel MKL implementation.

Arguments

max_cache_size

Max cache size (settings).

See also

[avx_math](#), [clr_math](#), [gpu_math](#), [sse_math](#)

microsoftml.mutualinformation_select: Feature selection based on mutual information

Article • 03/03/2023

Usage

```
microsoftml.mutualinformation_select(cols: [list, str], label: str,  
    num_features_to_keep: int = 1000, num_bins: int = 256, **kwargs)
```

Description

Selects the top k features across all specified columns ordered by their mutual information with the label column.

Details

The mutual information of two random variables X and Y is a measure of the mutual dependence between the variables. Formally, the mutual information can be written as:

$$I(X;Y) = E[\log(p(x,y)) - \log(p(x)) - \log(p(y))]$$

where the expectation is taken over the joint distribution of X and Y . Here $p(x,y)$ is the joint probability density function of X and Y , $p(x)$ and $p(y)$ are the marginal probability density functions of X and Y respectively. In general, a higher mutual information between the dependent variable (or label) and an independent variable (or feature) means that the label has higher mutual dependence over that feature.

The mutual information feature selection mode selects the features based on the mutual information. It keeps the top `num_features_to_keep` features with the largest mutual information with the label.

Arguments

cols

Specifies character string or list of the names of the variables to select.

label

Specifies the name of the label.

num_features_to_keep

If the number of features to keep is specified to be `n`, the transform picks the `n` features that have the highest mutual information with the dependent variable. The default value is 1000.

num_bins

Maximum number of bins for numerical values. Powers of 2 are recommended. The default value is 256.

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

[count_select](#)

References

[Wikipedia: Mutual Information](#) [↗](#)

microsoftml.n_gram: Converts text into features using n-grams

Article • 02/28/2023

Usage

```
microsoftml.n_gram(ngram_length: numbers.Real = 1,  
                   skip_length: numbers.Real = 0, all_lengths: bool = True,  
                   max_num_terms: list = [10000000], weighting: str = 'Tf')
```

Description

Extracts NGrams from text and convert them to vector using dictionary.

Arguments

ngram_length

Ngram length (settings).

skip_length

Maximum number of tokens to skip when constructing an ngram (settings).

all_lengths

Whether to include all ngram lengths up to NgramLength or only NgramLength (settings).

max_num_terms

Maximum number of ngrams to store in the dictionary (settings).

weighting

The weighting criteria (settings).

See also

[n_gram_hash](#), [featurize_text](#)

microsoftml.n_gram_hash: Converts text into features using hashed n-grams

Article • 03/03/2023

Usage

```
microsoftml.n_gram_hash(hash_bits: numbers.Real = 16,  
    ngram_length: numbers.Real = 1, skip_length: numbers.Real = 0,  
    all_lengths: bool = True, seed: numbers.Real = 314489979,  
    ordered: bool = True, invert_hash: numbers.Real = 0)
```

Description

Extracts NGrams from text and convert them to vector using hashing trick.

Arguments

hash_bits

Number of bits to hash into. Must be between 1 and 30, inclusive. (settings).

ngram_length

Ngram length (settings).

skip_length

Maximum number of tokens to skip when constructing an ngram (settings).

all_lengths

Whether to include all ngram lengths up to ngramLength or only ngramLength (settings).

seed

Hashing seed (settings).

ordered

Whether the position of each source column should be included in the hash (when there are multiple source columns). (settings).

invert_hash

Limit the number of keys used to generate the slot name to this many. 0 means no invert hashing, -1 means no limit. (settings).

See also

[n_gram](#), [featurize_text](#)

microsoftml.predefined: Removes predefined stopwords

Article • 02/28/2023

Usage

```
microsoftml.predefined()
```

Description

Remover with predefined list of stop words.

microsoftml.resize_image: Resizes an Image

Article • 03/03/2023

Usage

```
microsoftml.resize_image(cols: [str, dict, list], width: int = 224,  
    height: int = 224, resizing_option: ['IsoPad', 'IsoCrop',  
    'Aniso'] = 'IsoCrop', **kargs)
```

Description

Resizes an image to a specified dimension using a specified resizing method.

Details

`resize_image` resizes an image to the specified height and width using a specified resizing method. The input variables to this transform must be images, typically the result of the `load_image` transform.

Arguments

cols

A character string or list of variable names to transform. If `dict`, the keys represent the names of new variables to be created.

width

Specifies the width of the scaled image in pixels. The default value is 224.

height

Specifies the height of the scaled image in pixels. The default value is 224.

resizing_option

Specified the resizing method to use. Note that all methods are using bilinear interpolation. The options are:

- `"IsoPad"`: The image is resized such that the aspect ratio is preserved. If needed, the image is padded with black to fit the new width or height.
- `"IsoCrop"`: The image is resized such that the aspect ratio is preserved. If needed, the image is cropped to fit the new width or height.
- `"Aniso"`: The image is stretched to the new width and height, without preserving the aspect ratio.

The default value is `"IsoPad"`.

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

[load_image](#), [extract_pixels](#), [featurize_image](#).

Example

```
...
Example with images.
...
import numpy
import pandas
from microsoftml import rx_neural_network, rx_predict, rx_fast_linear
from microsoftml import load_image, resize_image, extract_pixels
from microsoftml.datasets.image import get_RevolutionAnalyticslogo

train = pandas.DataFrame(data=dict(Path=[get_RevolutionAnalyticslogo()],
Label=[True]))
```

```

# Loads the images from variable Path, resizes the images to 1x1 pixels
# and trains a neural net.
model1 = rx_neural_network("Label ~ Features", data=train,
    ml_transforms=[
        load_image(cols=dict(Features="Path")),
        resize_image(cols="Features", width=1, height=1,
resizing="Aniso"),
        extract_pixels(cols="Features")],
    ml_transform_vars=["Path"],
    num_hidden_nodes=1, num_iterations=1)

# Featurizes the images from variable Path using the default model, and
# trains a linear model on the result.
# If dnnModel == "AlexNet", the image has to be resized to 227x227.
model2 = rx_fast_linear("Label ~ Features ", data=train,
    ml_transforms=[
        load_image(cols=dict(Features="Path")),
        resize_image(cols="Features", width=224, height=224),
        extract_pixels(cols="Features")],
    ml_transform_vars=["Path"], max_iterations=1)

# We predict even if it does not make too much sense on this single image.
print("\nrx_neural_network")
prediction1 = rx_predict(model1, data=train)
print(prediction1)

print("\nrx_fast_linear")
prediction2 = rx_predict(model2, data=train)
print(prediction2)

```

Output:

```

Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Using: AVX Math

***** Net definition *****
input Data [3];
hidden H [1] sigmoid { // Depth 1
    from Data all;
}
output Result [1] sigmoid { // Depth 0

```

```

    from H all;
  }
**** End net definition ****
Input count: 3
Output count: 1
Output Function: Sigmoid
Loss Function: LogLoss
PreTrainer: NoPreTrainer

-----
Starting training...
Learning rate: 0.001000
Momentum: 0.000000
InitWtsDiameter: 0.100000

-----
Initializing 1 Hidden Layers, 6 Weights...
Estimated Pre-training MeanError = 0.707823
Iter:1/1, MeanErr=0.707823(0.00%), 0.01M WeightUpdates/sec
Done!
Estimated Post-training MeanError = 0.707499

-----
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.0820600
Elapsed time: 00:00:00.0090292
Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Using 2 threads to train.
Automatically choosing a check frequency of 2.
Auto-tuning parameters: L2 = 5.
Auto-tuning parameters: L1Threshold (L1/L2) = 1.
Using model from last iteration.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:01.0852660
Elapsed time: 00:00:00.0132126

rx_neural_network
Beginning processing data.
Rows Read: 1, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0441601
Finished writing 1 rows.
Writing completed.
  PredictedLabel      Score  Probability
0             False -0.028504    0.492875

rx_fast_linear
Beginning processing data.

```

Rows Read: 1, Read Time: 0.001, Transform Time: 0

Beginning processing data.

Elapsed time: 00:00:00.5196788

Finished writing 1 rows.

Writing completed.

	PredictedLabel	Score	Probability
0	False	0.0	0.5

microsoftml.rx_ensemble: Combine models into a single one

Article • 02/28/2023

Usage

```
microsoftml.rx_ensemble(formula: str,
    data: [<class 'revoscalepy.datasources.RxDataSource.RxDataSource'>,
    <class 'pandas.core.frame.DataFrame'>, <class 'list'>],
    trainers: typing.List[microsoftml.modules.base_learner.BaseLearner],
    method: str = None, model_count: int = None,
    random_seed: int = None, replace: bool = False,
    samp_rate: float = None, combine_method: ['Average', 'Median',
    'Vote'] = 'Median', max_calibration: int = 100000,
    split_data: bool = False, ml_transforms: list = None,
    ml_transform_vars: list = None, row_selection: str = None,
    transforms: dict = None, transform_objects: dict = None,
    transform_function: str = None,
    transform_variables: list = None,
    transform_packages: list = None,
    transform_environment: dict = None, blocks_per_read: int = None,
    report_progress: int = None, verbose: int = 1,
    compute_context:
    revoscalepy.computecontext.RxComputeContext.RxComputeContext = None)
```

Description

Train an ensemble of models.

Details

`rx_ensemble` is a function that trains a number of models of various kinds to obtain better predictive performance than could be obtained from a single model.

Arguments

formula

A symbolic or mathematical formula in valid Python syntax, enclosed in double quotes. A symbolic formula might reference objects in the data source, such as `"creditScore ~ yearsEmploy"`. Interaction terms (`creditScore * yearsEmploy`) and expressions (`creditScore == 1`) are not currently supported.

data

A data source object or a character string specifying a `.xdf` file or a data frame object. Alternatively, it can be a list of data sources indicating each model should be trained using one of the data sources in the list. In this case, the length of the data list must be equal to `model_count`.

trainers

A list of trainers with their arguments. The trainers are created by using `FastTrees`, `FastForest`, `FastLinear`, `LogisticRegression`, `NeuralNetwork`, or `OneClassSvm`.

method

A character string that specifies the type of ensemble: `"anomaly"` for Anomaly Detection, `"binary"` for Binary Classification, `multiClass` for Multiclass Classification, or `"regression"` for Regression.

random_seed

Specifies the random seed. The default value is `None`.

model_count

Specifies the number of models to train. If this number is greater than the length of the trainers list, the trainers list is duplicated to match `model_count`.

replace

A logical value specifying if the sampling of observations should be done with or without replacement. The default value is `False`.

samp_rate

A scalar of positive value specifying the percentage of observations to sample for each trainer. The default is `1.0` for sampling with replacement (i.e., `replace=True`) and `0.632` for sampling without replacement (i.e., `replace=False`). When `split_data` is `True`, the default of `samp_rate` is `1.0` (no sampling is done before splitting).

split_data

A logical value specifying whether or not to train the base models on non-overlapping partitions. The default is `False`. It is available only for `RxSpark` compute context and ignored for others.

combine_method

Specifies the method used to combine the models:

- `"Median"`: to compute the median of the individual model outputs,
- `"Average"`: to compute the average of the individual model outputs and
- `"Vote"`: to compute (pos-neg) / the total number of models, where 'pos' is the number of positive outputs and 'neg' is the number of negative outputs.

max_calibration

Specifies the maximum number of examples to use for calibration. This argument is ignored for all tasks other than binary classification.

ml_transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or *None* if no transforms are to be performed. Transforms that require an additional pass over the data (such as `featurize_text`, `categorical`) are not allowed. These transformations are performed after any specified R transformations. The default value is *None*.

ml_transform_vars

Specifies a character vector of variable names to be used in *ml_transforms* or *None* if none are to be used. The default value is *None*.

row_selection

NOT SUPPORTED. Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example:

- `rowSelection = "old"` will only use observations in which the value of the variable `old` is `True`.
- `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10.

The row selection is performed after processing any data transformations (see the arguments `transforms` or `transform_func`). As with all expressions, `row_selection` can be defined outside of the function call using the `expression` function.

transforms

NOT SUPPORTED. An expression of the form that represents the first round of variable transformations. As with all expressions, `transforms` (or `row_selection`) can be defined outside of the function call using the `expression` function.

transform_objects

NOT SUPPORTED. A named list that contains objects that can be referenced by `transforms`, `transform_function`, and `row_selection`.

transform_function

The variable transformation function.

transform_variables

A character vector of input data set variables needed for the transformation function.

transform_packages

NOT SUPPORTED. A character vector specifying additional Python packages (outside of those specified in `RxOptions.get_option("transform_packages")`) to be made available

and preloaded for use in variable transformation functions. For example, those explicitly defined in [revoscalepy](#) functions via their `transforms` and `transform_function` arguments or those defined implicitly via their `formula` or `row_selection` arguments. The `transform_packages` argument may also be *None*, indicating that no packages outside `RxOptions.get_option("transform_packages")` are preloaded.

transform_environment

NOT SUPPORTED. A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transform_environment = None`, a new "hash" environment with parent `revoscalepy.baseenv` is used instead.

blocks_per_read

Specifies the number of blocks to read for each chunk of data read from the data source.

report_progress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

compute_context

Sets the context in which computations are executed, specified with a valid `revoscalepy.RxComputeContext`. Currently local and [revoscalepy.RxSpark](#) compute contexts are supported. When [revoscalepy.RxSpark](#) is specified, the training of the

models is done in a distributed way, and the ensembling is done locally. Note that the compute context cannot be non-waiting.

Returns

A `rx_ensemble` object with the trained ensemble model.

microsoftml.rx_fast_forest: Random Forest

Article • 03/03/2023

Usage

```
microsoftml.rx_fast_forest(formula: str,  
    data: [revoscalepy.datasource.RxDataSource.RxDataSource,  
    pandas.core.frame.DataFrame], method: ['binary',  
    'regression'] = 'binary', num_trees: int = 100,  
    num_leaves: int = 20, min_split: int = 10,  
    example_fraction: float = 0.7, feature_fraction: float = 1,  
    split_fraction: float = 1, num_bins: int = 255,  
    first_use_penalty: float = 0, gain_conf_level: float = 0,  
    train_threads: int = 8, random_seed: int = None,  
    ml_transforms: list = None, ml_transform_vars: list = None,  
    row_selection: str = None, transforms: dict = None,  
    transform_objects: dict = None, transform_function: str = None,  
    transform_variables: list = None,  
    transform_packages: list = None,  
    transform_environment: dict = None, blocks_per_read: int = None,  
    report_progress: int = None, verbose: int = 1,  
    ensemble: microsoftml.modules.ensemble.EnsembleControl = None,  
    compute_context:  
    revoscalepy.computecontext.RxComputeContext.RxComputeContext = None)
```

Description

Machine Learning Fast Forest

Details

Decision trees are non-parametric models that perform a sequence of simple tests on inputs. This decision procedure maps them to outputs found in the training dataset whose inputs were similar to the instance being processed. A decision is made at each node of the binary tree data structure based on a measure of similarity that maps each instance recursively through the branches of the tree until the appropriate leaf node is reached and the output decision returned.

Decision trees have several advantages:

- They are efficient in both computation and memory usage during training and prediction.
- They can represent non-linear decision boundaries.
- They perform integrated feature selection and classification.
- They are resilient in the presence of noisy features.

Fast forest regression is a random forest and quantile regression forest implementation using the regression tree learner in [rx_fast_trees](#). The model consists of an ensemble of decision trees. Each tree in a decision forest outputs a Gaussian distribution by way of prediction. An aggregation is performed over the ensemble of trees to find a Gaussian distribution closest to the combined distribution for all trees in the model.

This decision forest classifier consists of an ensemble of decision trees. Generally, ensemble models provide better coverage and accuracy than single decision trees. Each tree in a decision forest outputs a Gaussian distribution.

Arguments

formula

The formula as described in [revoscalepy.rx_formula](#). Interaction terms and `F()` are not currently supported in [microsoftml](#).

data

A data source object or a character string specifying a *.xdf* file or a data frame object.

method

A character string denoting Fast Tree type:

- `"binary"` for the default Fast Tree Binary Classification or
- `"regression"` for Fast Tree Regression.

num_trees

Specifies the total number of decision trees to create in the ensemble. By creating more decision trees, you can potentially get better coverage, but the training time increases.

The default value is 100.

num_leaves

The maximum number of leaves (terminal nodes) that can be created in any tree. Higher values potentially increase the size of the tree and get better precision, but risk overfitting and requiring longer training times. The default value is 20.

min_split

Minimum number of training instances required to form a leaf. That is, the minimal number of documents allowed in a leaf of a regression tree, out of the sub-sampled data. A 'split' means that features in each level of the tree (node) are randomly divided. The default value is 10.

example_fraction

The fraction of randomly chosen instances to use for each tree. The default value is 0.7.

feature_fraction

The fraction of randomly chosen features to use for each tree. The default value is 0.7.

split_fraction

The fraction of randomly chosen features to use on each split. The default value is 0.7.

num_bins

Maximum number of distinct values (bins) per feature. The default value is 255.

first_use_penalty

The feature first use penalty coefficient. The default value is 0.

gain_conf_level

Tree fitting gain confidence requirement (should be in the range `[0,1]`). The default value is 0.

train_threads

The number of threads to use in training. If *None* is specified, the number of threads to use is determined internally. The default value is *None*.

random_seed

Specifies the random seed. The default value is *None*.

ml_transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or *None* if no transforms are to be performed. See [featurize_text](#), [categorical](#), and [categorical_hash](#), for transformations that are supported. These transformations are performed after any specified Python transformations. The default value is *None*.

ml_transform_vars

Specifies a character vector of variable names to be used in `ml_transforms` or *None* if none are to be used. The default value is *None*.

row_selection

NOT SUPPORTED. Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example:

- `row_selection = "old"` will only use observations in which the value of the variable `old` is `True`.
- `row_selection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10.

The row selection is performed after processing any data transformations (see the arguments `transforms` or `transform_function`). As with all expressions, `row_selection` can be defined outside of the function call using the `expression` function.

transforms

NOT SUPPORTED. An expression of the form that represents the first round of variable transformations. As with all expressions, `transforms` (or `row_selection`) can be defined outside of the function call using the `expression` function.

transform_objects

NOT SUPPORTED. A named list that contains objects that can be referenced by `transforms`, `transform_function`, and `row_selection`.

transform_function

The variable transformation function.

transform_variables

A character vector of input data set variables needed for the transformation function.

transform_packages

NOT SUPPORTED. A character vector specifying additional Python packages (outside of those specified in `RxOptions.get_option("transform_packages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in `revoscalepy` functions via their `transforms` and `transform_function` arguments or those defined implicitly via their `formula` or `row_selection` arguments. The `transform_packages` argument may also be `None`, indicating that no packages outside `RxOptions.get_option("transform_packages")` are preloaded.

transform_environment

NOT SUPPORTED. A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transform_environment = None`, a new "hash" environment with parent `revoscalepy.baseenv` is used instead.

blocks_per_read

Specifies the number of blocks to read for each chunk of data read from the data source.

report_progress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

compute_context

Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxInSqlServer` compute contexts are supported.

ensemble

Control parameters for ensembling.

Returns

A `FastForest` object with the trained model.

Note

This algorithm is multi-threaded and will always attempt to load the entire dataset into memory.

See also

[rx_fast_trees](#), [rx_predict](#)

References

[Wikipedia: Random forest](#) ↗

[Quantile regression forest](#) ↗

[From Stumps to Trees to Forests](#)

Binary classification example

```
...
Binary Classification.
...

import numpy
import pandas
from microsoftml import rx_fast_forest, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

infert = get_dataset("infert")

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

infertdf = infert.as_df()
infertdf["isCase"] = infertdf.case == 1
data_train, data_test, y_train, y_test = train_test_split(infertdf,
infertdf.isCase)

forest_model = rx_fast_forest(
    formula=" isCase ~ age + parity + education + spontaneous + induced ",
    data=data_train)

# RuntimeError: The type (RxTextData) for file is not supported.
score_ds = rx_predict(forest_model, data=data_test,
    extra_vars_to_write=["isCase", "Score"])

# Print the first five rows
print(rx_data_step(score_ds, number_rows_read=5))
```

Output:

```
Not adding a normalizer.
Making per-feature arrays
Changing data from row-wise to column-wise
```

```

Beginning processing data.
Rows Read: 186, Read Time: 0, Transform Time: 0
Beginning processing data.
Processed 186 instances
Binning and forming Feature objects
Reserved memory for tree learner: 7176 bytes
Starting to train ...
Not training a calibrator because a valid calibrator trainer was not
provided.
Elapsed time: 00:00:00.2704185
Elapsed time: 00:00:00.0443884
Beginning processing data.
Rows Read: 62, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0253862
Finished writing 62 rows.
Writing completed.
Rows Read: 5, Total Rows Processed: 5, Total Chunk Time: Less than .001
seconds
  isCase PredictedLabel      Score
0  False              False -36.205067
1   True              False -40.396084
2  False              False -33.242531
3  False              False -87.212494
4   True              False -13.100666

```

Regression example

```

...
Regression.
...

import numpy
import pandas
from microsoftml import rx_fast_forest, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

airquality = get_dataset("airquality")

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

airquality = airquality.as_df()

#####
# Estimate a regression fast forest

```

```

# Use the built-in data set 'airquality' to create test and train data

df = airquality[airquality.Ozone.notnull()]
df["Ozone"] = df.Ozone.astype(float)

data_train, data_test, y_train, y_test = train_test_split(df, df.Ozone)

airFormula = " Ozone ~ Solar_R + Wind + Temp "

# Regression Fast Forest for train data
ff_reg = rx_fast_forest(airFormula, method="regression", data=data_train)

# Put score and model variables in data frame
score_df = rx_predict(ff_reg, data=data_test, write_model_vars=True)
print(score_df.head())

# Plot actual versus predicted values with smoothed line
# Supported in the next version.
# rx_line_plot(" Score ~ Ozone ", type=["p", "smooth"], data=score_df)

```

Output:

```

Not adding a normalizer.
Making per-feature arrays
Changing data from row-wise to column-wise
Beginning processing data.
Rows Read: 87, Read Time: 0, Transform Time: 0
Beginning processing data.
Warning: Skipped 4 instances with missing features during training
Processed 83 instances
Binning and forming Feature objects
Reserved memory for tree learner: 21372 bytes
Starting to train ...
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.0644269
Elapsed time: 00:00:00.0109290
Beginning processing data.
Rows Read: 29, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0314390
Finished writing 29 rows.
Writing completed.

```

	Solar_R	Wind	Temp	Score
0	190.0	7.4	67.0	26.296144
1	20.0	16.6	63.0	14.274153
2	320.0	16.6	73.0	23.421144
3	187.0	5.1	87.0	80.662109
4	175.0	7.4	89.0	67.570549

microsoftml.rx_fast_linear: Linear Model with Stochastic Dual Coordinate Ascent

Article • 03/03/2023

Usage

```
microsoftml.rx_fast_linear()
```

Description

A Stochastic Dual Coordinate Ascent (SDCA) optimization trainer for linear binary classification and regression.

Details

`rx_fast_linear` is a trainer based on the Stochastic Dual Coordinate Ascent (SDCA) method, a state-of-the-art optimization technique for convex objective functions. The algorithm can be scaled for use on large out-of-memory data sets due to a semi-asynchronized implementation that supports multi-threading. Convergence is underwritten by periodically enforcing synchronization between primal and dual updates in a separate thread. Several choices of loss functions are also provided. The SDCA method combines several of the best properties and capabilities of logistic regression and SVM algorithms. For more information on SDCA, see the citations in the reference section.

Traditional optimization algorithms, such as stochastic gradient descent (SGD), optimize the empirical loss function directly. The SDCA chooses a different approach that optimizes the dual problem instead. The dual loss function is parametrized by per-example weights. In each iteration, when a training example from the training data set is read, the corresponding example weight is adjusted so that the dual loss function is optimized with respect to the current example. No learning rate is needed by SDCA to determine step size as is required by various gradient descent methods.

`rx_fast_linear` supports binary classification with three types of loss functions currently: Log loss, hinge loss, and smoothed hinge loss. Linear regression also supports with squared loss function. Elastic net regularization can be specified by the `l2_weight`

and `l1_weight` parameters. Note that the `l2_weight` has an effect on the rate of convergence. In general, the larger the `l2_weight`, the faster SDCA converges.

Note that `rx_fast_linear` is a stochastic and streaming optimization algorithm. The results depend on the order of the training data. For reproducible results, it is recommended that one sets `shuffle` to `False` and `train_threads` to `1`.

Arguments

formula

The formula described in `revoscalepy.rx_formula`. Interaction terms and `F()` are not currently supported in [microsoftml](#).

data

A data source object or a character string specifying a `.xdf` file or a data frame object.

method

Specifies the model type with a character string: `"binary"` for the default binary classification or `"regression"` for linear regression.

loss_function

Specifies the empirical loss function to optimize. For binary classification, the following choices are available:

- `log_loss`: The log-loss. This is the default.
- `hinge_loss`: The SVM hinge loss. Its parameter represents the margin size.
- `smooth_hinge_loss`: The smoothed hinge loss. Its parameter represents the smoothing constant.

For linear regression, squared loss `squared_loss` is currently supported. When this parameter is set to `None`, its default value depends on the type of learning:

- `log_loss` for binary classification.
- `squared_loss` for linear regression.

The following example changes the `loss_function` to `hinge_loss`: `rx_fast_linear(..., loss_function=hinge_loss())`.

l1_weight

Specifies the L1 regularization weight. The value must be either non-negative or *None*. If *None* is specified, the actual value is automatically computed based on data set. *None* is the default value.

l2_weight

Specifies the L2 regularization weight. The value must be either non-negative or *None*. If *None* is specified, the actual value is automatically computed based on data set. *None* is the default value.

train_threads

Specifies how many concurrent threads can be used to run the algorithm. When this parameter is set to *None*, the number of threads used is determined based on the number of logical processors available to the process as well as the sparsity of data. Set it to `1` to run the algorithm in a single thread.

convergence_tolerance

Specifies the tolerance threshold used as a convergence criterion. It must be between 0 and 1. The default value is `0.1`. The algorithm is considered to have converged if the relative duality gap, which is the ratio between the duality gap and the primal loss, falls below the specified convergence tolerance.

max_iterations

Specifies an upper bound on the number of training iterations. This parameter must be positive or *None*. If *None* is specified, the actual value is automatically computed based on data set. Each iteration requires a complete pass over the training data. Training terminates after the total number of iterations reaches the specified upper bound or when the loss function converges, whichever happens earlier.

shuffle

Specifies whether to shuffle the training data. Set `True` to shuffle the data; `False` not to shuffle. The default value is `True`. SDCA is a stochastic optimization algorithm. If shuffling is turned on, the training data is shuffled on each iteration.

check_frequency

The number of iterations after which the loss function is computed and checked to determine whether it has converged. The value specified must be a positive integer or `None`. If `None`, the actual value is automatically computed based on data set. Otherwise, for example, if `checkFrequency = 5` is specified, then the loss function is computed and convergence is checked every 5 iterations. The computation of the loss function requires a separate complete pass over the training data.

normalize

Specifies the type of automatic normalization used:

- `"Auto"`: if normalization is needed, it is performed automatically. This is the default choice.
- `"No"`: no normalization is performed.
- `"Yes"`: normalization is performed.
- `"Warn"`: if normalization is needed, a warning message is displayed, but normalization is not performed.

Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MaxMin` normalizer is used. It normalizes values in an interval $[a, b]$ where $-1 \leq a \leq 0$ and $0 \leq b \leq 1$ and $b - a = 1$. This normalizer preserves sparsity by mapping zero to zero.

ml_transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or `None` if no transforms are to be performed. See [featurize_text](#), [categorical](#), and [categorical_hash](#), for transformations that are supported. These transformations are performed after any specified Python transformations. The default value is `None`.

ml_transform_vars

Specifies a character vector of variable names to be used in `ml_transforms` or `None` if none are to be used. The default value is `None`.

row_selection

NOT SUPPORTED. Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example:

- `row_selection = "old"` will only use observations in which the value of the variable `old` is `True`.
- `row_selection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10.

The row selection is performed after processing any data transformations (see the arguments `transforms` or `transform_function`). As with all expressions, `row_selection` can be defined outside of the function call using the `expression` function.

transforms

NOT SUPPORTED. An expression of the form that represents the first round of variable transformations. As with all expressions, `transforms` (or `row_selection`) can be defined outside of the function call using the `expression` function.

transform_objects

NOT SUPPORTED. A named list that contains objects that can be referenced by `transforms`, `transform_function`, and `row_selection`.

transform_function

The variable transformation function.

transform_variables

A character vector of input data set variables needed for the transformation function.

transform_packages

NOT SUPPORTED. A character vector specifying additional Python packages (outside of those specified in `RxOptions.get_option("transform_packages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in `revoscalepy` functions via their `transforms` and `transform_function` arguments or those defined implicitly via their `formula` or `row_selection` arguments. The `transform_packages` argument may also be `None`, indicating that no packages outside `RxOptions.get_option("transform_packages")` are preloaded.

transform_environment

NOT SUPPORTED. A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transform_environment = None`, a new "hash" environment with parent `revoscalepy.baseenv` is used instead.

blocks_per_read

Specifies the number of blocks to read for each chunk of data read from the data source.

report_progress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

compute_context

Sets the context in which computations are executed, specified with a valid `revoscalepy.RxComputeContext`. Currently local and [revoscalepy.RxInSqlServer](#) compute contexts are supported.

ensemble

Control parameters for ensembling.

Returns

A [FastLinear](#) object with the trained model.

Note

This algorithm is multi-threaded and will not attempt to load the entire dataset into memory.

See also

[hinge_loss](#), [log_loss](#), [smoothed_hinge_loss](#), [squared_loss](#), [rx_predict](#)

References

[Scaling Up Stochastic Dual Coordinate Ascent](#) ↗

[Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization](#) ↗

Binary classification example

```
...
Binary Classification.
...
import numpy
import pandas
from microsoftml import rx_fast_linear, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

infert = get_dataset("infert")
```

```

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

infertdf = infert.as_df()
infertdf["isCase"] = infertdf.case == 1
data_train, data_test, y_train, y_test = train_test_split(infertdf,
infertdf.isCase)

forest_model = rx_fast_linear(
    formula=" isCase ~ age + parity + education + spontaneous + induced ",
    data=data_train)

# RuntimeError: The type (RxTextData) for file is not supported.
score_ds = rx_predict(forest_model, data=data_test,
    extra_vars_to_write=["isCase", "Score"])

# Print the first five rows
print(rx_data_step(score_ds, number_rows_read=5))

```

Output:

```

Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 186, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 186, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 186, Read Time: 0, Transform Time: 0
Beginning processing data.
Using 2 threads to train.
Automatically choosing a check frequency of 2.
Auto-tuning parameters: maxIterations = 8064.
Auto-tuning parameters: L2 = 2.666837E-05.
Auto-tuning parameters: L1Threshold (L1/L2) = 0.
Using best model from iteration 568.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.5810985
Elapsed time: 00:00:00.0084876
Beginning processing data.
Rows Read: 62, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0292334
Finished writing 62 rows.
Writing completed.
Rows Read: 5, Total Rows Processed: 5, Total Chunk Time: Less than .001

```

seconds	isCase	PredictedLabel	Score	Probability
0	True	True	0.990544	0.729195
1	False	False	-2.307120	0.090535
2	False	False	-0.608565	0.352387
3	True	True	1.028217	0.736570
4	True	False	-3.913066	0.019588

Regression example

```

...
Regression.
...
import numpy
import pandas
from microsoftml import rx_fast_linear, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

attitude = get_dataset("attitude")

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

attitudedf = attitude.as_df()
data_train, data_test = train_test_split(attitudedf)

model = rx_fast_linear(
    formula="rating ~ complaints + privileges + learning + raises + critical
+ advance",
    method="regression",
    data=data_train)

# RuntimeError: The type (RxTextData) for file is not supported.
score_ds = rx_predict(model, data=data_test,
                      extra_vars_to_write=["rating"])

# Print the first five rows
print(rx_data_step(score_ds, number_rows_read=5))

```

Output:

Automatically adding a MinMax normalization transform, use 'norm=Warn' or 'norm=No' to turn this behavior off.

Beginning processing data.

Rows Read: 22, Read Time: 0.001, Transform Time: 0

Beginning processing data.

Beginning processing data.

Rows Read: 22, Read Time: 0.001, Transform Time: 0

Beginning processing data.

Beginning processing data.

Rows Read: 22, Read Time: 0, Transform Time: 0

Beginning processing data.

Using 2 threads to train.

Automatically choosing a check frequency of 2.

Auto-tuning parameters: maxIterations = 68180.

Auto-tuning parameters: L2 = 0.01.

Auto-tuning parameters: L1Threshold (L1/L2) = 0.

Using best model from iteration 54.

Not training a calibrator because it is not needed.

Elapsed time: 00:00:00.1114324

Elapsed time: 00:00:00.0090901

Beginning processing data.

Rows Read: 8, Read Time: 0, Transform Time: 0

Beginning processing data.

Elapsed time: 00:00:00.0330772

Finished writing 8 rows.

Writing completed.

Rows Read: 5, Total Rows Processed: 5, Total Chunk Time: Less than .001 seconds

	rating	Score
0	71.0	72.630440
1	67.0	56.995350
2	67.0	52.958641
3	72.0	80.894539
4	50.0	38.375427

loss functions

- [*microsoftml.hinge_loss*](#): Hinge loss function
- [*microsoftml.log_loss*](#): Log loss function
- [*microsoftml.smoothed_hinge_loss*](#): Smoothed hinge loss function
- [*microsoftml.squared_loss*](#): Squared loss function

microsoftml.rx_fast_trees: Boosted Trees

Article • 03/03/2023

Usage

```
microsoftml.rx_fast_trees(formula: str,  
    data: [revoscalepy.datasources.RxDataSource.RxDataSource,  
    pandas.core.frame.DataFrame], method: ['binary',  
    'regression'] = 'binary', num_trees: int = 100,  
    num_leaves: int = 20, learning_rate: float = 0.2,  
    min_split: int = 10, example_fraction: float = 0.7,  
    feature_fraction: float = 1, split_fraction: float = 1,  
    num_bins: int = 255, first_use_penalty: float = 0,  
    gain_conf_level: float = 0, unbalanced_sets: bool = False,  
    train_threads: int = 8, random_seed: int = None,  
    ml_transforms: list = None, ml_transform_vars: list = None,  
    row_selection: str = None, transforms: dict = None,  
    transform_objects: dict = None, transform_function: str = None,  
    transform_variables: list = None,  
    transform_packages: list = None,  
    transform_environment: dict = None, blocks_per_read: int = None,  
    report_progress: int = None, verbose: int = 1,  
    ensemble: microsoftml.modules.ensemble.EnsembleControl = None,  
    compute_context:  
    revoscalepy.computecontext.RxComputeContext.RxComputeContext = None)
```

Description

Machine Learning Fast Tree

Details

`rx_fast_trees` is an implementation of FastRank. FastRank is an efficient implementation of the MART gradient boosting algorithm. Gradient boosting is a machine learning technique for regression problems. It builds each regression tree in a step-wise fashion, using a predefined loss function to measure the error for each step and corrects for it in the next. So this prediction model is actually an ensemble of weaker prediction models. In regression problems, boosting builds a series of such trees in a step-wise fashion and then selects the optimal tree using an arbitrary differentiable loss function.

MART learns an ensemble of regression trees, which is a decision tree with scalar values in its leaves. A decision (or regression) tree is a binary tree-like flow chart, where at each interior node one decides which of the two child nodes to continue to based on one of the feature values from the input. At each leaf node, a value is returned. In the interior nodes, the decision is based on the test " $x \leq v$ ", where x is the value of the feature in the input sample and v is one of the possible values of this feature. The functions that can be produced by a regression tree are all the piece-wise constant functions.

The ensemble of trees is produced by computing, in each step, a regression tree that approximates the gradient of the loss function, and adding it to the previous tree with coefficients that minimize the loss of the new tree. The output of the ensemble produced by MART on a given instance is the sum of the tree outputs.

- In case of a binary classification problem, the output is converted to a probability by using some form of calibration.
- In case of a regression problem, the output is the predicted value of the function.
- In case of a ranking problem, the instances are ordered by the output value of the ensemble.

If `method` is set to `"regression"`, a regression version of FastTree is used. If set to `"ranking"`, a ranking version of FastTree is used. In the ranking case, the instances should be ordered by the output of the tree ensemble. The only difference in the settings of these versions is in the calibration settings, which are needed only for classification.

Arguments

formula

The formula as described in `revoScalePy.rx_formula`. Interaction terms and `F()` are not currently supported in [microsoftml](#).

data

A data source object or a character string specifying a `.xdf` file or a data frame object.

method

A character string that specifies the type of Fast Tree: "binary" for the default Fast Tree Binary Classification or "regression" for Fast Tree Regression.

num_trees

Specifies the total number of decision trees to create in the ensemble. By creating more decision trees, you can potentially get better coverage, but the training time increases. The default value is 100.

num_leaves

The maximum number of leaves (terminal nodes) that can be created in any tree. Higher values potentially increase the size of the tree and get better precision, but risk overfitting and requiring longer training times. The default value is 20.

learning_rate

Determines the size of the step taken in the direction of the gradient in each step of the learning process. This determines how fast or slow the learner converges on the optimal solution. If the step size is too big, you might overshoot the optimal solution. If the step size is too small, training takes longer to converge to the best solution.

min_split

Minimum number of training instances required to form a leaf. That is, the minimal number of documents allowed in a leaf of a regression tree, out of the sub-sampled data. A 'split' means that features in each level of the tree (node) are randomly divided. The default value is 10. Only the number of instances is counted even if instances are weighted.

example_fraction

The fraction of randomly chosen instances to use for each tree. The default value is 0.7.

feature_fraction

The fraction of randomly chosen features to use for each tree. The default value is 1.

split_fraction

The fraction of randomly chosen features to use on each split. The default value is 1.

num_bins

Maximum number of distinct values (bins) per feature. If the feature has fewer values than the number indicated, each value is placed in its own bin. If there are more values, the algorithm creates `numBins` bins.

first_use_penalty

The feature first use penalty coefficient. This is a form of regularization that incurs a penalty for using a new feature when creating the tree. Increase this value to create trees that don't use many features. The default value is 0.

gain_conf_level

Tree fitting gain confidence requirement (should be in the range [0,1)). The default value is 0.

unbalanced_sets

If `True`, derivatives optimized for unbalanced sets are used. Only applicable when `type` equal to `"binary"`. The default value is `False`.

train_threads

The number of threads to use in training. The default value is 8.

random_seed

Specifies the random seed. The default value is *None*.

ml_transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or *None* if no transforms are to be performed. See [featurize_text](#), [categorical](#), and [categorical_hash](#), for transformations that are supported. These transformations are performed after any specified Python transformations. The default value is *None*.

ml_transform_vars

Specifies a character vector of variable names to be used in `ml_transforms` or `None` if none are to be used. The default value is `None`.

row_selection

NOT SUPPORTED. Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example:

- `row_selection = "old"` will only use observations in which the value of the variable `old` is `True`.
- `row_selection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10.

The row selection is performed after processing any data transformations (see the arguments `transforms` or `transform_function`). As with all expressions, `row_selection` can be defined outside of the function call using the `expression` function.

transforms

NOT SUPPORTED. An expression of the form that represents the first round of variable transformations. As with all expressions, `transforms` (or `row_selection`) can be defined outside of the function call using the `expression` function.

transform_objects

NOT SUPPORTED. A named list that contains objects that can be referenced by `transforms`, `transform_function`, and `row_selection`.

transform_function

The variable transformation function.

transform_variables

A character vector of input data set variables needed for the transformation function.

transform_packages

NOT SUPPORTED. A character vector specifying additional Python packages (outside of those specified in `RxOptions.get_option("transform_packages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in `revoscalepy` functions via their `transforms` and `transform_function` arguments or those defined implicitly via their `formula` or `row_selection` arguments. The `transform_packages` argument may also be `None`, indicating that no packages outside `RxOptions.get_option("transform_packages")` are preloaded.

transform_environment

NOT SUPPORTED. A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transform_environment = None`, a new "hash" environment with parent `revoscalepy.baseenv` is used instead.

blocks_per_read

Specifies the number of blocks to read for each chunk of data read from the data source.

report_progress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

compute_context

Sets the context in which computations are executed, specified with a valid `revoscalepy.RxComputeContext`. Currently local and [revoscalepy.RxInSqlServer](#) compute contexts are supported.

ensemble

Control parameters for ensembling.

Returns

A [FastTrees](#) object with the trained model.

Note

This algorithm is multi-threaded and will always attempt to load the entire dataset into memory.

See also

[rx_fast_forest](#), [rx_predict](#)

References

[Wikipedia: Gradient boosting \(Gradient tree boosting\)](#) [↗](#)

[Greedy function approximation: A gradient boosting machine.](#) [↗](#)

Binary Classification example

```
...
Binary Classification.
...
import numpy
import pandas
from microsoftml import rx_fast_trees, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

infert = get_dataset("infert")
```

```

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

infertdf = infert.as_df()
infertdf["isCase"] = infertdf.case == 1
data_train, data_test, y_train, y_test = train_test_split(infertdf,
infertdf.isCase)

trees_model = rx_fast_trees(
    formula=" isCase ~ age + parity + education + spontaneous + induced ",
    data=data_train)

# RuntimeError: The type (RxTextData) for file is not supported.
score_ds = rx_predict(trees_model, data=data_test,
    extra_vars_to_write=["isCase", "Score"])

# Print the first five rows
print(rx_data_step(score_ds, number_rows_read=5))

```

Output:

```

Not adding a normalizer.
Making per-feature arrays
Changing data from row-wise to column-wise
Beginning processing data.
Rows Read: 186, Read Time: 0, Transform Time: 0
Beginning processing data.
Processed 186 instances
Binning and forming Feature objects
Reserved memory for tree learner: 7020 bytes
Starting to train ...
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.0949161
Elapsed time: 00:00:00.0112103
Beginning processing data.
Rows Read: 62, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0230457
Finished writing 62 rows.
Writing completed.
Rows Read: 5, Total Rows Processed: 5, Total Chunk Time: 0.001 seconds

```

	isCase	PredictedLabel	Score	Probability
0	False	False	-4.722279	0.131369
1	False	False	-11.550012	0.009757
2	False	False	-7.312314	0.050935
3	True	True	3.889991	0.825778
4	False	False	-6.361800	0.072782

Regression example

```
...
Regression.
...

import numpy
import pandas
from microsoftml import rx_fast_trees, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

airquality = get_dataset("airquality")

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

airquality = airquality.as_df()

#####
# Estimate a regression fast forest
# Use the built-in data set 'airquality' to create test and train data

df = airquality[airquality.Ozone.notnull()]
df["Ozone"] = df.Ozone.astype(float)

data_train, data_test, y_train, y_test = train_test_split(df, df.Ozone)

airFormula = " Ozone ~ Solar_R + Wind + Temp "

# Regression Fast Forest for train data
ff_reg = rx_fast_trees(airFormula, method="regression", data=data_train)

# Put score and model variables in data frame
score_df = rx_predict(ff_reg, data=data_test, write_model_vars=True)
print(score_df.head())

# Plot actual versus predicted values with smoothed line
# Supported in the next version.
# rx_line_plot(" Score ~ Ozone ", type=["p", "smooth"], data=score_df)
```

Output:

```
'unbalanced_sets' ignored for method 'regression'
Not adding a normalizer.
```

```
Making per-feature arrays
Changing data from row-wise to column-wise
Beginning processing data.
Rows Read: 87, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Warning: Skipped 4 instances with missing features during training
Processed 83 instances
Binning and forming Feature objects
Reserved memory for tree learner: 21528 bytes
Starting to train ...
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.0512720
Elapsed time: 00:00:00.0094435
Beginning processing data.
Rows Read: 29, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0229873
Finished writing 29 rows.
Writing completed.
```

	Solar_R	Wind	Temp	Score
0	115.0	7.4	76.0	26.003876
1	307.0	12.0	66.0	18.057747
2	230.0	10.9	75.0	10.896211
3	259.0	9.7	73.0	13.726607
4	92.0	15.5	84.0	37.972855

microsoftml.rx_featurize: Data transformation for data sources

Article • 03/03/2023

Usage

```
microsoftml.rx_featurize(data:
typing.Union[revoscalepy.datasources.RxDataSource.RxDataSource,
pandas.core.frame.DataFrame],
output_data:
typing.Union[revoscalepy.datasources.RxDataSource.RxDataSource,
str] = None, overwrite: bool = False,
data_threads: int = None, random_seed: int = None,
max_slots: int = 5000, ml_transforms: list = None,
ml_transform_vars: list = None, row_selection: str = None,
transforms: dict = None, transform_objects: dict = None,
transform_function: str = None,
transform_variables: list = None,
transform_packages: list = None,
transform_environment: dict = None, blocks_per_read: int = None,
report_progress: int = None, verbose: int = 1,
compute_context:
revoscalepy.computecontext.RxComputeContext.RxComputeContext = None)
```

Description

Transforms data from an input data set to an output data set.

Arguments

data

A [revoscalepy](#) data source object, a data frame, or the path to a `.xdf` file.

output_data

Output text or xdf file name or an `RxDataSource` with write capabilities in which to store transformed data. If *None*, a data frame is returned. The default value is *None*.

overwrite

If `True`, an existing `output_data` is overwritten; if `False` an existing `output_data` is not overwritten. The default value is `False`.

data_threads

An integer specifying the desired degree of parallelism in the data pipeline. If `None`, the number of threads used is determined internally. The default value is `None`.

random_seed

Specifies the random seed. The default value is `None`.

max_slots

Max slots to return for vector valued columns (≤ 0 to return all).

ml_transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or `None` if no transforms are to be performed. See [featurize_text](#), [categorical](#), and [categorical_hash](#), for transformations that are supported. These transformations are performed after any specified Python transformations. The default value is `None`.

ml_transform_vars

Specifies a character vector of variable names to be used in `ml_transforms` or `None` if none are to be used. The default value is `None`.

row_selection

NOT SUPPORTED. Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example:

- `row_selection = "old"` will only use observations in which the value of the variable `old` is `True`.

- `row_selection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10.

The row selection is performed after processing any data transformations (see the arguments `transforms` or `transform_function`). As with all expressions, `row_selection` can be defined outside of the function call using the `expression` function.

transforms

NOT SUPPORTED. An expression of the form that represents the first round of variable transformations. As with all expressions, `transforms` (or `row_selection`) can be defined outside of the function call using the `expression` function. The default value is *None*.

transform_objects

NOT SUPPORTED. A named list that contains objects that can be referenced by `transforms`, `transform_function`, and `row_selection`. The default value is *None*.

transform_function

The variable transformation function. The default value is *None*.

transform_variables

A character vector of input data set variables needed for the transformation function. The default value is *None*.

transform_packages

NOT SUPPORTED. A character vector specifying additional Python packages (outside of those specified in `RxOptions.get_option("transform_packages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in [revoscalepy](#) functions via their `transforms` and `transform_function` arguments or those defined implicitly via their `formula` or `row_selection` arguments. The `transform_packages` argument may also be *None*, indicating that no packages outside `RxOptions.get_option("transform_packages")` are preloaded.

transform_environment

NOT SUPPORTED. A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transform_environment = None`, a new "hash" environment with parent `revoscalepy.baseenv` is used instead. The default value is *None*.

blocks_per_read

Specifies the number of blocks to read for each chunk of data read from the data source.

report_progress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

The default value is `1`.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information. The default value is `1`.

compute_context

Sets the context in which computations are executed, specified with a valid `revoscalepy.RxComputeContext`. Currently `local` and `revoscalepy.RxInSqlServer` compute contexts are supported.

Returns

A data frame or an `revoscalepy.RxDataSource` object representing the created output data.

See also

[rx_predict](#), [revoscalepy.rx_data_step](#), [revoscalepy.rx_import](#).

Example

```
...
Example with rx_featurize.
...
import numpy
import pandas
from microsoftml import rx_featurize, categorical

# rx_featurize basically allows you to access data from the MicrosoftML
transforms
# In this example we'll look at getting the output of the categorical
transform
# Create the data
categorical_data = pandas.DataFrame(data=dict(places_visited=[
    "London", "Brunei", "London", "Paris", "Seria"]),
    dtype="category")

print(categorical_data)

# Invoke the categorical transform
categorized = rx_featurize(data=categorical_data,
                           ml_transforms=
[categorical(cols=dict(xdatacat="places_visited"))])

# Now let's look at the data
print(categorized)
```

Output:

```
  places_visited
0           London
1           Brunei
2           London
3            Paris
4            Seria
Beginning processing data.
Rows Read: 5, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 5, Read Time: 0, Transform Time: 0
Beginning processing data.
```

Elapsed time: 00:00:00.0521300

Finished writing 5 rows.

Writing completed.

	places_visited	xdatacat.London	xdatacat.Brunei	xdatacat.Paris	\
0	London	1.0	0.0	0.0	
1	Brunei	0.0	1.0	0.0	
2	London	1.0	0.0	0.0	
3	Paris	0.0	0.0	1.0	
4	Seria	0.0	0.0	0.0	

	xdatacat.Seria
0	0.0
1	0.0
2	0.0
3	0.0
4	1.0

microsoftml.rx_logistic_regression: Logistic Regression

Article • 03/03/2023

Usage

```
microsoftml.rx_logistic_regression(formula: str,  
    data: [revoscalepy.datasources.RxDataSource.RxDataSource,  
    pandas.core.frame.DataFrame], method: ['binary',  
    'multiClass'] = 'binary', l2_weight: float = 1,  
    l1_weight: float = 1, opt_tol: float = 1e-07,  
    memory_size: int = 20, init_wts_diameter: float = 0,  
    max_iterations: int = 2147483647,  
    show_training_stats: bool = False, sgd_init_tol: float = 0,  
    train_threads: int = None, dense_optimizer: bool = False,  
    normalize: ['No', 'Warn', 'Auto', 'Yes'] = 'Auto',  
    ml_transforms: list = None, ml_transform_vars: list = None,  
    row_selection: str = None, transforms: dict = None,  
    transform_objects: dict = None, transform_function: str = None,  
    transform_variables: list = None,  
    transform_packages: list = None,  
    transform_environment: dict = None, blocks_per_read: int = None,  
    report_progress: int = None, verbose: int = 1,  
    ensemble: microsoftml.modules.ensemble.EnsembleControl = None,  
    compute_context:  
    revoscalepy.computecontext.RxComputeContext.RxComputeContext = None)
```

Description

Machine Learning Logistic Regression

Details

Logistic Regression is a classification method used to predict the value of a categorical dependent variable from its relationship to one or more independent variables assumed to have a logistic distribution. If the dependent variable has only two possible values (success/failure), then the logistic regression is binary. If the dependent variable has more than two possible values (blood type given diagnostic test results), then the logistic regression is multinomial.

The optimization technique used for `rx_logistic_regression` is the limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS). Both the L-BFGS and regular BFGS algorithms use quasi-Newtonian methods to estimate the computationally intensive Hessian matrix in the equation used by Newton's method to calculate steps. But the L-BFGS approximation uses only a limited amount of memory to compute the next step direction, so that it is especially suited for problems with a large number of variables. The `memory_size` parameter specifies the number of past positions and gradients to store for use in the computation of the next step.

This learner can use elastic net regularization: a linear combination of L1 (lasso) and L2 (ridge) regularizations. Regularization is a method that can render an ill-posed problem more tractable by imposing constraints that provide information to supplement the data and that prevents overfitting by penalizing models with extreme coefficient values. This can improve the generalization of the model learned by selecting the optimal complexity in the bias-variance tradeoff. Regularization works by adding the penalty that is associated with coefficient values to the error of the hypothesis. An accurate model with extreme coefficient values would be penalized more, but a less accurate model with more conservative values would be penalized less. L1 and L2 regularization have different effects and uses that are complementary in certain respects.

- `l1_weight`: can be applied to sparse models, when working with high-dimensional data. It pulls small weights associated features that are relatively unimportant towards 0.
- `l2_weight`: is preferable for data that is not sparse. It pulls large weights towards zero.

Adding the ridge penalty to the regularization overcomes some of lasso's limitations. It can improve its predictive accuracy, for example, when the number of predictors is greater than the sample size. If `x = l1_weight` and `y = l2_weight`, `ax + by = c` defines the linear span of the regularization terms. The default values of `x` and `y` are both `1`. An aggressive regularization can harm predictive capacity by excluding important variables out of the model. So choosing the optimal values for the regularization parameters is important for the performance of the logistic regression model.

Arguments

formula

The formula as described in `revoscalepy.rx_formula` Interaction terms and `F()` are not currently supported in [microsoftml](#).

data

A data source object or a character string specifying a *.xdf* file or a data frame object.

method

A character string that specifies the type of Logistic Regression: "binary" for the default binary classification logistic regression or "multiClass" for multinomial logistic regression.

l2_weight

The L2 regularization weight. Its value must be greater than or equal to 0 and the default value is set to 1.

l1_weight

The L1 regularization weight. Its value must be greater than or equal to 0 and the default value is set to 1.

opt_tol

Threshold value for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model. Smaller values are slower, but more accurate. The default value is $1e-07$.

memory_size

Memory size for L-BFGS, specifying the number of past positions and gradients to store for the computation of the next step. This optimization parameter limits the amount of memory that is used to compute the magnitude and direction of the next step. When you specify less memory, training is faster but less accurate. Must be greater than or equal to 1 and the default value is 20.

max_iterations

Sets the maximum number of iterations. After this number of steps, the algorithm stops even if it has not satisfied convergence criteria.

show_training_stats

Specify `True` to show the statistics of training data and the trained model; otherwise, `False`. The default value is `False`. For additional information about model statistics, see `summary.ml_model()`.

sgd_init_tol

Set to a number greater than 0 to use Stochastic Gradient Descent (SGD) to find the initial parameters. A non-zero value set specifies the tolerance SGD uses to determine convergence. The default value is `0` specifying that SGD is not used.

init_wts_diameter

Sets the initial weights diameter that specifies the range from which values are drawn for the initial weights. These weights are initialized randomly from within this range. For example, if the diameter is specified to be `d`, then the weights are uniformly distributed between `-d/2` and `d/2`. The default value is `0`, which specifies that all the weights are initialized to `0`.

train_threads

The number of threads to use in training the model. This should be set to the number of cores on the machine. Note that L-BFGS multi-threading attempts to load dataset into memory. In case of out-of-memory issues, set `train_threads` to `1` to turn off multi-threading. If `None` the number of threads to use is determined internally. The default value is `None`.

dense_optimizer

If `True`, forces densification of the internal optimization vectors. If `False`, enables the logistic regression optimizer use sparse or dense internal states as it finds appropriate. Setting `denseOptimizer` to `True` requires the internal optimizer to use a dense internal state, which may help alleviate load on the garbage collector for some varieties of larger problems.

normalize

Specifies the type of automatic normalization used:

- `"Auto"`: if normalization is needed, it is performed automatically. This is the default choice.
- `"No"`: no normalization is performed.
- `"Yes"`: normalization is performed.
- `"Warn"`: if normalization is needed, a warning message is displayed, but normalization is not performed.

Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MinMax` normalizer is used. It normalizes values in an interval $[a, b]$ where $-1 \leq a \leq 0$ and $0 \leq b \leq 1$ and $b - a = 1$. This normalizer preserves sparsity by mapping zero to zero.

ml_transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or *None* if no transforms are to be performed. See [featurize_text](#), [categorical](#), and [categorical_hash](#), for transformations that are supported. These transformations are performed after any specified Python transformations. The default value is *None*.

ml_transform_vars

Specifies a character vector of variable names to be used in `ml_transforms` or *None* if none are to be used. The default value is *None*.

row_selection

NOT SUPPORTED. Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example:

- `row_selection = "old"` will only use observations in which the value of the variable `old` is `True`.
- `row_selection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10.

The row selection is performed after processing any data transformations (see the arguments `transforms` or `transform_function`). As with all expressions, `row_selection` can be defined outside of the function call using the `expression` function.

transforms

NOT SUPPORTED. An expression of the form that represents the first round of variable transformations. As with all expressions, `transforms` (or `row_selection`) can be defined outside of the function call using the `expression` function.

transform_objects

NOT SUPPORTED. A named list that contains objects that can be referenced by `transforms`, `transform_function`, and `row_selection`.

transform_function

The variable transformation function.

transform_variables

A character vector of input data set variables needed for the transformation function.

transform_packages

NOT SUPPORTED. A character vector specifying additional Python packages (outside of those specified in `RxOptions.get_option("transform_packages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in [revoscalepy](#) functions via their `transforms` and `transform_function` arguments or those defined implicitly via their `formula` or `row_selection` arguments. The `transform_packages` argument may also be *None*, indicating that no packages outside `RxOptions.get_option("transform_packages")` are preloaded.

transform_environment

NOT SUPPORTED. A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transform_environment = None`, a new "hash" environment with parent `revoscalepy.baseenv` is used instead.

blocks_per_read

Specifies the number of blocks to read for each chunk of data read from the data source.

report_progress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

compute_context

Sets the context in which computations are executed, specified with a valid `revoscalepy.RxComputeContext`. Currently `local` and `revoscalepy.RxInSqlServer` compute contexts are supported.

ensemble

Control parameters for ensembling.

Returns

A `LogisticRegression` object with the trained model.

Note

This algorithm will attempt to load the entire dataset into memory when `train_threads > 1` (multi-threading).

See also

[rx_predict](#)

References

[Wikipedia: L-BFGS](#)

[Wikipedia: Logistic regression](#)

[Scalable Training of L1-Regularized Log-Linear Models](#)

[Test Run - L1 and L2 Regularization for Machine Learning](#)

Binary classification example

```
...
Binary Classification.
...

import numpy
import pandas
from microsoftml import rx_logistic_regression, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

infert = get_dataset("infert")

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

infertdf = infert.as_df()
infertdf["isCase"] = infertdf.case == 1
data_train, data_test, y_train, y_test = train_test_split(infertdf,
infertdf.isCase)

model = rx_logistic_regression(
    formula=" isCase ~ age + parity + education + spontaneous + induced ",
    data=data_train)
```

```

print(model.coef_)

# RuntimeError: The type (RxTextData) for file is not supported.
score_ds = rx_predict(model, data=data_test,
                      extra_vars_to_write=["isCase", "Score"])

# Print the first five rows
print(rx_data_step(score_ds, number_rows_read=5))

```

Output:

```

Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 186, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 186, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 186, Read Time: 0, Transform Time: 0
Beginning processing data.
LBFGS multi-threading will attempt to load dataset into memory. In case of
out-of-memory issues, turn off multi-threading by setting trainThreads to 1.
Beginning optimization
num vars: 6
improvement criterion: Mean Improvement
L1 regularization selected 5 of 6 weights.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.0646405
Elapsed time: 00:00:00.0083991
OrderedDict([('Bias)', -1.2366217374801636), ('spontaneous',
1.9391206502914429), ('induced', 0.7497404217720032), ('parity',
-0.31517016887664795), ('age', -3.162723260174971e-06)])
Beginning processing data.
Rows Read: 62, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0287290
Finished writing 62 rows.
Writing completed.
Rows Read: 5, Total Rows Processed: 5, Total Chunk Time: 0.001 seconds
  isCase PredictedLabel      Score  Probability
0  False                False -1.341681    0.207234
1   True                 True  0.597440    0.645070
2  False                 True  0.544912    0.632954
3  False                False -1.289152    0.215996
4  False                False -1.019339    0.265156

```

MultiClass classification example

```
...
MultiClass Classification
...
import numpy
import pandas
from microsoftml import rx_logistic_regression, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

iris = get_dataset("iris")

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

irisdf = iris.as_df()
irisdf["Species"] = irisdf["Species"].astype("category")
data_train, data_test, y_train, y_test = train_test_split(irisdf,
irisdf.Species)

model = rx_logistic_regression(
    formula=" Species ~ Sepal_Length + Sepal_Width + Petal_Length +
Petal_Width ",
    method="multiClass",
    data=data_train)

print(model.coef_)

# RuntimeError: The type (RxTextData) for file is not supported.
score_ds = rx_predict(model, data=data_test,
    extra_vars_to_write=["Species", "Score"])

# Print the first five rows
print(rx_data_step(score_ds, number_rows_read=5))
```

Output:

```
Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 112, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 112, Read Time: 0, Transform Time: 0
```

```

Beginning processing data.
Beginning processing data.
Rows Read: 112, Read Time: 0, Transform Time: 0
Beginning processing data.
LBFGS multi-threading will attempt to load dataset into memory. In case of
out-of-memory issues, turn off multi-threading by setting trainThreads to 1.
Beginning optimization
num vars: 15
improvement criterion: Mean Improvement
L1 regularization selected 9 of 15 weights.
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.0493224
Elapsed time: 00:00:00.0080558
OrderedDict([('setosa+(Bias)', 2.074636697769165), ('versicolor+(Bias)',
0.4899507164955139), ('virginica+(Bias)', -2.564580202102661),
('setosa+Petal_Width', -2.8389241695404053), ('setosa+Petal_Length',
-2.4824044704437256), ('setosa+Sepal_Width', 0.274869441986084),
('versicolor+Sepal_Width', -0.2645561397075653), ('virginica+Petal_Width',
2.6924400329589844), ('virginica+Petal_Length', 1.5976412296295166)])
Beginning processing data.
Rows Read: 38, Read Time: 0, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0331861
Finished writing 38 rows.
Writing completed.
Rows Read: 5, Total Rows Processed: 5, Total Chunk Time: 0.001 seconds
  Species  Score.0  Score.1  Score.2
0  virginica  0.044230  0.364927  0.590843
1    setosa  0.767412  0.210586  0.022002
2    setosa  0.756523  0.221933  0.021543
3    setosa  0.767652  0.211191  0.021157
4  versicolor  0.116369  0.498615  0.385016

```


microsoftml.rx_neural_network: Neural Network

Article • 03/03/2023

Usage

```
microsoftml.rx_neural_network(formula: str,
    data: [revoscalepy.datasource.RxDataSource.RxDataSource,
    pandas.core.frame.DataFrame], method: ['binary', 'multiClass',
    'regression'] = 'binary', num_hidden_nodes: int = 100,
    num_iterations: int = 100,
    optimizer: [<function adadelta_optimizer at 0x0000007156EAC048>,
    <function sgd_optimizer at 0x0000007156E9FB70>] = {'Name':
'SgdOptimizer',
    'Settings': {}}, net_definition: str = None,
    init_wts_diameter: float = 0.1, max_norm: float = 0,
    acceleration: [<function avx_math at 0x0000007156E9FEA0>,
    <function clr_math at 0x0000007156EAC158>,
    <function gpu_math at 0x0000007156EAC1E0>,
    <function mkl_math at 0x0000007156EAC268>,
    <function sse_math at 0x0000007156EAC2F0>] = {'Name': 'AvxMath',
    'Settings': {}}, mini_batch_size: int = 1, normalize: ['No',
    'Warn', 'Auto', 'Yes'] = 'Auto', ml_transforms: list = None,
    ml_transform_vars: list = None, row_selection: str = None,
    transforms: dict = None, transform_objects: dict = None,
    transform_function: str = None,
    transform_variables: list = None,
    transform_packages: list = None,
    transform_environment: dict = None, blocks_per_read: int = None,
    report_progress: int = None, verbose: int = 1,
    ensemble: microsoftml.modules.ensemble.EnsembleControl = None,
    compute_context:
    revoscalepy.computecontext.RxComputeContext.RxComputeContext = None)
```

Description

Neural networks for regression modeling and for Binary and multi-class classification.

Details

A neural network is a class of prediction models inspired by the human brain. A neural network can be represented as a weighted directed graph. Each node in the graph is

called a neuron. The neurons in the graph are arranged in layers, where neurons in one layer are connected by a weighted edge (weights can be 0 or positive numbers) to neurons in the next layer. The first layer is called the input layer, and each neuron in the input layer corresponds to one of the features. The last layer of the function is called the output layer. So in the case of binary neural networks it contains two output neurons, one for each class, whose values are the probabilities of belonging to each class. The remaining layers are called hidden layers. The values of the neurons in the hidden layers and in the output layer are set by calculating the weighted sum of the values of the neurons in the previous layer and applying an activation function to that weighted sum. A neural network model is defined by the structure of its graph (namely, the number of hidden layers and the number of neurons in each hidden layer), the choice of activation function, and the weights on the graph edges. The neural network algorithm tries to learn the optimal weights on the edges based on the training data.

Although neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are also easily adapted to regression problems. Any class of statistical models can be considered a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Neural network regression is especially suited to problems where a more traditional regression model cannot fit a solution.

Arguments

formula

The formula as described in `revoscalepy.rx_formula`. Interaction terms and `F()` are not currently supported in [microsoftml](#).

data

A data source object or a character string specifying a `.xdf` file or a data frame object.

method

A character string denoting Fast Tree type:

- `"binary"` for the default binary classification neural network.
- `"multiClass"` for multi-class classification neural network.
- `"regression"` for a regression neural network.

num_hidden_nodes

The default number of hidden nodes in the neural net. The default value is 100.

num_iterations

The number of iterations on the full training set. The default value is 100.

optimizer

A list specifying either the `sgd` or `adaptive` optimization algorithm. This list can be created using `sgd_optimizer` or `adadelat_optimizer`. The default value is `sgd`.

net_definition

The Net# definition of the structure of the neural network. For more information about the Net# language, see [Reference Guide](#)

init_wts_diameter

Sets the initial weights diameter that specifies the range from which values are drawn for the initial learning weights. The weights are initialized randomly from within this range. The default value is 0.1.

max_norm

Specifies an upper bound to constrain the norm of the incoming weight vector at each hidden unit. This can be very important in max out neural networks as well as in cases where training produces unbounded weights.

acceleration

Specifies the type of hardware acceleration to use. Possible values are "sse_math" and "gpu_math". For GPU acceleration, it is recommended to use a miniBatchSize greater than one. If you want to use the GPU acceleration, there are additional manual setup steps are required:

- Download and install NVidia CUDA Toolkit 6.5 ([CUDA Toolkit](#)).
- Download and install NVidia cuDNN v2 Library ([cudnn Library](#)).

- Find the libs directory of the microsoftml package by calling `import microsoftml, os, os.path.join(microsoftml.__path__[0], "mxLibs")`.
- Copy `cublas64_65.dll`, `cuda64_65.dll` and `cusparse64_65.dll` from the CUDA Toolkit 6.5 into the libs directory of the microsoftml package.
- Copy `cuda64_65.dll` from the cuDNN v2 Library into the libs directory of the microsoftml package.

mini_batch_size

Sets the mini-batch size. Recommended values are between 1 and 256. This parameter is only used when the acceleration is GPU. Setting this parameter to a higher value improves the speed of training, but it might negatively affect the accuracy. The default value is 1.

normalize

Specifies the type of automatic normalization used:

- `"Warn"`: if normalization is needed, it is performed automatically. This is the default choice.
- `"No"`: no normalization is performed.
- `"Yes"`: normalization is performed.
- `"Auto"`: if normalization is needed, a warning message is displayed, but normalization is not performed.

Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MinMax` normalizer is used. It normalizes values in an interval $[a, b]$ where $-1 \leq a \leq 0$ and $0 \leq b \leq 1$ and $b - a = 1$. This normalizer preserves sparsity by mapping zero to zero.

ml_transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or `None` if no transforms are to be performed. See [featurize_text](#), [categorical](#), and

`categorical_hash`, for transformations that are supported. These transformations are performed after any specified Python transformations. The default value is *None*.

`ml_transform_vars`

Specifies a character vector of variable names to be used in `ml_transforms` or *None* if none are to be used. The default value is *None*.

`row_selection`

NOT SUPPORTED. Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example:

- `row_selection = "old"` will only use observations in which the value of the variable `old` is `True`.
- `row_selection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10.

The row selection is performed after processing any data transformations (see the arguments `transforms` or `transform_function`). As with all expressions, `row_selection` can be defined outside of the function call using the `expression` function.

`transforms`

NOT SUPPORTED. An expression of the form that represents the first round of variable transformations. As with all expressions, `transforms` (or `row_selection`) can be defined outside of the function call using the `expression` function.

`transform_objects`

NOT SUPPORTED. A named list that contains objects that can be referenced by `transforms`, `transform_function`, and `row_selection`.

`transform_function`

The variable transformation function.

transform_variables

A character vector of input data set variables needed for the transformation function.

transform_packages

NOT SUPPORTED. A character vector specifying additional Python packages (outside of those specified in `RxOptions.get_option("transform_packages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in `revoscalepy` functions via their `transforms` and `transform_function` arguments or those defined implicitly via their `formula` or `row_selection` arguments. The `transform_packages` argument may also be `None`, indicating that no packages outside `RxOptions.get_option("transform_packages")` are preloaded.

transform_environment

NOT SUPPORTED. A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transform_environment = None`, a new "hash" environment with parent `revoscalepy.baseenv` is used instead.

blocks_per_read

Specifies the number of blocks to read for each chunk of data read from the data source.

report_progress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of

information.

compute_context

Sets the context in which computations are executed, specified with a valid `revoscalepy.RxComputeContext`. Currently local and [revoscalepy.RxInSqlServer](#) compute contexts are supported.

ensemble

Control parameters for ensembling.

Returns

A [NeuralNetwork](#) object with the trained model.

Note

This algorithm is single-threaded and will not attempt to load the entire dataset into memory.

See also

[adadelta_optimizer](#), [sgd_optimizer](#), [avx_math](#), [clr_math](#), [gpu_math](#), [mkl_math](#), [sse_math](#), [rx_predict](#).

References

[Wikipedia: Artificial neural network](#) [↗](#)

Binary classification example

```
...  
Binary Classification.  
...  
import numpy  
import pandas  
from microsoftml import rx_neural_network, rx_predict  
from revoscalepy.etl.RxDataStep import rx_data_step
```

```

from microsoftml.datasets.datasets import get_dataset

infert = get_dataset("infert")

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

infertdf = infert.as_df()
infertdf["isCase"] = infertdf.case == 1
data_train, data_test, y_train, y_test = train_test_split(infertdf,
infertdf.isCase)

forest_model = rx_neural_network(
    formula=" isCase ~ age + parity + education + spontaneous + induced ",
    data=data_train)

# RuntimeError: The type (RxTextData) for file is not supported.
score_ds = rx_predict(forest_model, data=data_test,
                      extra_vars_to_write=["isCase", "Score"])

# Print the first five rows
print(rx_data_step(score_ds, number_rows_read=5))

```

Output:

```

Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 186, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 186, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 186, Read Time: 0, Transform Time: 0
Beginning processing data.
Using: AVX Math

***** Net definition *****
input Data [5];
hidden H [100] sigmoid { // Depth 1
    from Data all;
}
output Result [1] sigmoid { // Depth 0
    from H all;
}
***** End net definition *****
Input count: 5

```


Output count: 1
Output Function: Sigmoid
Loss Function: LogLoss
PreTrainer: NoPreTrainer

Starting training...
Learning rate: 0.001000
Momentum: 0.000000
InitWtsDiameter: 0.100000

Initializing 1 Hidden Layers, 701 Weights...

Estimated Pre-training MeanError = 0.742343

Iter:1/100, MeanErr=0.680245(-8.37%), 119.87M WeightUpdates/sec
Iter:2/100, MeanErr=0.637843(-6.23%), 122.52M WeightUpdates/sec
Iter:3/100, MeanErr=0.635404(-0.38%), 122.24M WeightUpdates/sec
Iter:4/100, MeanErr=0.634980(-0.07%), 73.36M WeightUpdates/sec
Iter:5/100, MeanErr=0.635287(0.05%), 128.26M WeightUpdates/sec
Iter:6/100, MeanErr=0.634572(-0.11%), 131.05M WeightUpdates/sec
Iter:7/100, MeanErr=0.634827(0.04%), 124.27M WeightUpdates/sec
Iter:8/100, MeanErr=0.635359(0.08%), 123.69M WeightUpdates/sec
Iter:9/100, MeanErr=0.635244(-0.02%), 119.35M WeightUpdates/sec
Iter:10/100, MeanErr=0.634712(-0.08%), 127.80M WeightUpdates/sec
Iter:11/100, MeanErr=0.635105(0.06%), 122.69M WeightUpdates/sec
Iter:12/100, MeanErr=0.635226(0.02%), 98.61M WeightUpdates/sec
Iter:13/100, MeanErr=0.634977(-0.04%), 127.88M WeightUpdates/sec
Iter:14/100, MeanErr=0.634347(-0.10%), 123.25M WeightUpdates/sec
Iter:15/100, MeanErr=0.634891(0.09%), 124.27M WeightUpdates/sec
Iter:16/100, MeanErr=0.635116(0.04%), 123.06M WeightUpdates/sec
Iter:17/100, MeanErr=0.633770(-0.21%), 122.05M WeightUpdates/sec
Iter:18/100, MeanErr=0.634992(0.19%), 128.79M WeightUpdates/sec
Iter:19/100, MeanErr=0.634385(-0.10%), 122.95M WeightUpdates/sec
Iter:20/100, MeanErr=0.634752(0.06%), 127.14M WeightUpdates/sec
Iter:21/100, MeanErr=0.635043(0.05%), 123.44M WeightUpdates/sec
Iter:22/100, MeanErr=0.634845(-0.03%), 121.81M WeightUpdates/sec
Iter:23/100, MeanErr=0.634850(0.00%), 125.11M WeightUpdates/sec
Iter:24/100, MeanErr=0.634617(-0.04%), 122.18M WeightUpdates/sec
Iter:25/100, MeanErr=0.634675(0.01%), 125.69M WeightUpdates/sec
Iter:26/100, MeanErr=0.634911(0.04%), 122.44M WeightUpdates/sec
Iter:27/100, MeanErr=0.634311(-0.09%), 121.90M WeightUpdates/sec
Iter:28/100, MeanErr=0.634798(0.08%), 123.54M WeightUpdates/sec
Iter:29/100, MeanErr=0.634674(-0.02%), 127.53M WeightUpdates/sec
Iter:30/100, MeanErr=0.634546(-0.02%), 100.96M WeightUpdates/sec
Iter:31/100, MeanErr=0.634859(0.05%), 124.40M WeightUpdates/sec
Iter:32/100, MeanErr=0.634747(-0.02%), 128.21M WeightUpdates/sec
Iter:33/100, MeanErr=0.634842(0.02%), 125.82M WeightUpdates/sec
Iter:34/100, MeanErr=0.634703(-0.02%), 77.48M WeightUpdates/sec
Iter:35/100, MeanErr=0.634804(0.02%), 122.21M WeightUpdates/sec
Iter:36/100, MeanErr=0.634690(-0.02%), 112.48M WeightUpdates/sec
Iter:37/100, MeanErr=0.634654(-0.01%), 119.18M WeightUpdates/sec
Iter:38/100, MeanErr=0.634885(0.04%), 137.19M WeightUpdates/sec
Iter:39/100, MeanErr=0.634723(-0.03%), 113.80M WeightUpdates/sec
Iter:40/100, MeanErr=0.634714(0.00%), 127.50M WeightUpdates/sec
Iter:41/100, MeanErr=0.634794(0.01%), 129.54M WeightUpdates/sec
Iter:42/100, MeanErr=0.633835(-0.15%), 133.05M WeightUpdates/sec
Iter:43/100, MeanErr=0.634401(0.09%), 128.95M WeightUpdates/sec

Iter:44/100, MeanErr=0.634575(0.03%), 123.42M WeightUpdates/sec
Iter:45/100, MeanErr=0.634673(0.02%), 123.78M WeightUpdates/sec
Iter:46/100, MeanErr=0.634692(0.00%), 119.04M WeightUpdates/sec
Iter:47/100, MeanErr=0.634476(-0.03%), 122.95M WeightUpdates/sec
Iter:48/100, MeanErr=0.634583(0.02%), 97.87M WeightUpdates/sec
Iter:49/100, MeanErr=0.634706(0.02%), 121.41M WeightUpdates/sec
Iter:50/100, MeanErr=0.634564(-0.02%), 120.58M WeightUpdates/sec
Iter:51/100, MeanErr=0.634118(-0.07%), 120.17M WeightUpdates/sec
Iter:52/100, MeanErr=0.634699(0.09%), 127.27M WeightUpdates/sec
Iter:53/100, MeanErr=0.634123(-0.09%), 110.51M WeightUpdates/sec
Iter:54/100, MeanErr=0.634390(0.04%), 123.74M WeightUpdates/sec
Iter:55/100, MeanErr=0.634461(0.01%), 113.66M WeightUpdates/sec
Iter:56/100, MeanErr=0.634415(-0.01%), 118.61M WeightUpdates/sec
Iter:57/100, MeanErr=0.634453(0.01%), 114.99M WeightUpdates/sec
Iter:58/100, MeanErr=0.634478(0.00%), 104.53M WeightUpdates/sec
Iter:59/100, MeanErr=0.634010(-0.07%), 124.62M WeightUpdates/sec
Iter:60/100, MeanErr=0.633901(-0.02%), 118.93M WeightUpdates/sec
Iter:61/100, MeanErr=0.634088(0.03%), 40.46M WeightUpdates/sec
Iter:62/100, MeanErr=0.634046(-0.01%), 94.65M WeightUpdates/sec
Iter:63/100, MeanErr=0.634233(0.03%), 27.18M WeightUpdates/sec
Iter:64/100, MeanErr=0.634596(0.06%), 123.94M WeightUpdates/sec
Iter:65/100, MeanErr=0.634185(-0.06%), 125.01M WeightUpdates/sec
Iter:66/100, MeanErr=0.634469(0.04%), 119.41M WeightUpdates/sec
Iter:67/100, MeanErr=0.634333(-0.02%), 124.11M WeightUpdates/sec
Iter:68/100, MeanErr=0.634203(-0.02%), 112.68M WeightUpdates/sec
Iter:69/100, MeanErr=0.633854(-0.05%), 118.62M WeightUpdates/sec
Iter:70/100, MeanErr=0.634319(0.07%), 123.59M WeightUpdates/sec
Iter:71/100, MeanErr=0.634423(0.02%), 122.51M WeightUpdates/sec
Iter:72/100, MeanErr=0.634388(-0.01%), 126.15M WeightUpdates/sec
Iter:73/100, MeanErr=0.634230(-0.02%), 126.51M WeightUpdates/sec
Iter:74/100, MeanErr=0.634011(-0.03%), 128.32M WeightUpdates/sec
Iter:75/100, MeanErr=0.634294(0.04%), 127.48M WeightUpdates/sec
Iter:76/100, MeanErr=0.634372(0.01%), 123.51M WeightUpdates/sec
Iter:77/100, MeanErr=0.632020(-0.37%), 122.12M WeightUpdates/sec
Iter:78/100, MeanErr=0.633770(0.28%), 119.55M WeightUpdates/sec
Iter:79/100, MeanErr=0.633504(-0.04%), 124.21M WeightUpdates/sec
Iter:80/100, MeanErr=0.634154(0.10%), 125.94M WeightUpdates/sec
Iter:81/100, MeanErr=0.633491(-0.10%), 120.83M WeightUpdates/sec
Iter:82/100, MeanErr=0.634212(0.11%), 128.60M WeightUpdates/sec
Iter:83/100, MeanErr=0.634138(-0.01%), 73.58M WeightUpdates/sec
Iter:84/100, MeanErr=0.634244(0.02%), 124.08M WeightUpdates/sec
Iter:85/100, MeanErr=0.634065(-0.03%), 96.43M WeightUpdates/sec
Iter:86/100, MeanErr=0.634174(0.02%), 124.28M WeightUpdates/sec
Iter:87/100, MeanErr=0.633966(-0.03%), 125.24M WeightUpdates/sec
Iter:88/100, MeanErr=0.633989(0.00%), 130.31M WeightUpdates/sec
Iter:89/100, MeanErr=0.633767(-0.04%), 115.73M WeightUpdates/sec
Iter:90/100, MeanErr=0.633831(0.01%), 122.81M WeightUpdates/sec
Iter:91/100, MeanErr=0.633219(-0.10%), 114.91M WeightUpdates/sec
Iter:92/100, MeanErr=0.633589(0.06%), 93.29M WeightUpdates/sec
Iter:93/100, MeanErr=0.634086(0.08%), 123.31M WeightUpdates/sec
Iter:94/100, MeanErr=0.634075(0.00%), 120.99M WeightUpdates/sec
Iter:95/100, MeanErr=0.634071(0.00%), 122.49M WeightUpdates/sec
Iter:96/100, MeanErr=0.633523(-0.09%), 116.48M WeightUpdates/sec
Iter:97/100, MeanErr=0.634103(0.09%), 128.85M WeightUpdates/sec
Iter:98/100, MeanErr=0.633836(-0.04%), 123.87M WeightUpdates/sec

```
Iter:99/100, MeanErr=0.633772(-0.01%), 128.17M WeightUpdates/sec  
Iter:100/100, MeanErr=0.633684(-0.01%), 123.65M WeightUpdates/sec  
Done!
```

```
Estimated Post-training MeanError = 0.631268
```

```
Not training a calibrator because it is not needed.
```

```
Elapsed time: 00:00:00.2454094
```

```
Elapsed time: 00:00:00.0082325
```

```
Beginning processing data.
```

```
Rows Read: 62, Read Time: 0.001, Transform Time: 0
```

```
Beginning processing data.
```

```
Elapsed time: 00:00:00.0297006
```

```
Finished writing 62 rows.
```

```
Writing completed.
```

```
Rows Read: 5, Total Rows Processed: 5, Total Chunk Time: 0.001 seconds
```

	isCase	PredictedLabel	Score	Probability
0	True	False	-0.689636	0.334114
1	True	False	-0.710219	0.329551
2	True	False	-0.712912	0.328956
3	False	False	-0.700765	0.331643
4	True	False	-0.689783	0.334081

MultiClass classification example

```
...  
MultiClass Classification.  
...  
import numpy  
import pandas  
from microsoftml import rx_neural_network, rx_predict  
from revoscalepy.etl.RxDataStep import rx_data_step  
from microsoftml.datasets.datasets import get_dataset  
  
iris = get_dataset("iris")  
  
import sklearn  
if sklearn.__version__ < "0.18":  
    from sklearn.cross_validation import train_test_split  
else:  
    from sklearn.model_selection import train_test_split  
  
irisdf = iris.as_df()  
irisdf["Species"] = irisdf["Species"].astype("category")  
data_train, data_test, y_train, y_test = train_test_split(irisdf,  
irisdf.Species)  
  
model = rx_neural_network(  
    formula=" Species ~ Sepal_Length + Sepal_Width + Petal_Length +  
Petal_Width ",  
    method="multiClass",
```

```

data=data_train)

# RuntimeError: The type (RxTextData) for file is not supported.
score_ds = rx_predict(model, data=data_test,
                      extra_vars_to_write=["Species", "Score"])

# Print the first five rows
print(rx_data_step(score_ds, number_rows_read=5))

```

Output:

```

Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 112, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 112, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 112, Read Time: 0, Transform Time: 0
Beginning processing data.
Using: AVX Math

***** Net definition *****
  input Data [4];
  hidden H [100] sigmoid { // Depth 1
    from Data all;
  }
  output Result [3] softmax { // Depth 0
    from H all;
  }
***** End net definition *****
Input count: 4
Output count: 3
Output Function: SoftMax
Loss Function: LogLoss
PreTrainer: NoPreTrainer

-----
Starting training...
Learning rate: 0.001000
Momentum: 0.000000
InitWtsDiameter: 0.100000

-----
Initializing 1 Hidden Layers, 803 Weights...
Estimated Pre-training MeanError = 1.949606
Iter:1/100, MeanErr=1.937924(-0.60%), 98.43M WeightUpdates/sec
Iter:2/100, MeanErr=1.921153(-0.87%), 96.21M WeightUpdates/sec
Iter:3/100, MeanErr=1.920000(-0.06%), 95.55M WeightUpdates/sec
Iter:4/100, MeanErr=1.917267(-0.14%), 81.25M WeightUpdates/sec
Iter:5/100, MeanErr=1.917611(0.02%), 102.44M WeightUpdates/sec

```

Iter:6/100, MeanErr=1.918476(0.05%), 106.16M WeightUpdates/sec
Iter:7/100, MeanErr=1.916096(-0.12%), 97.85M WeightUpdates/sec
Iter:8/100, MeanErr=1.919486(0.18%), 77.99M WeightUpdates/sec
Iter:9/100, MeanErr=1.916452(-0.16%), 95.67M WeightUpdates/sec
Iter:10/100, MeanErr=1.916024(-0.02%), 102.06M WeightUpdates/sec
Iter:11/100, MeanErr=1.917155(0.06%), 99.21M WeightUpdates/sec
Iter:12/100, MeanErr=1.918543(0.07%), 99.25M WeightUpdates/sec
Iter:13/100, MeanErr=1.919120(0.03%), 85.38M WeightUpdates/sec
Iter:14/100, MeanErr=1.917713(-0.07%), 103.00M WeightUpdates/sec
Iter:15/100, MeanErr=1.917675(0.00%), 98.70M WeightUpdates/sec
Iter:16/100, MeanErr=1.917982(0.02%), 99.10M WeightUpdates/sec
Iter:17/100, MeanErr=1.916254(-0.09%), 103.41M WeightUpdates/sec
Iter:18/100, MeanErr=1.915691(-0.03%), 102.00M WeightUpdates/sec
Iter:19/100, MeanErr=1.914844(-0.04%), 86.64M WeightUpdates/sec
Iter:20/100, MeanErr=1.919268(0.23%), 94.68M WeightUpdates/sec
Iter:21/100, MeanErr=1.918748(-0.03%), 108.11M WeightUpdates/sec
Iter:22/100, MeanErr=1.917997(-0.04%), 96.33M WeightUpdates/sec
Iter:23/100, MeanErr=1.914987(-0.16%), 82.84M WeightUpdates/sec
Iter:24/100, MeanErr=1.916550(0.08%), 99.70M WeightUpdates/sec
Iter:25/100, MeanErr=1.915401(-0.06%), 96.69M WeightUpdates/sec
Iter:26/100, MeanErr=1.916092(0.04%), 101.62M WeightUpdates/sec
Iter:27/100, MeanErr=1.916381(0.02%), 98.81M WeightUpdates/sec
Iter:28/100, MeanErr=1.917414(0.05%), 102.29M WeightUpdates/sec
Iter:29/100, MeanErr=1.917316(-0.01%), 100.17M WeightUpdates/sec
Iter:30/100, MeanErr=1.916507(-0.04%), 82.09M WeightUpdates/sec
Iter:31/100, MeanErr=1.915786(-0.04%), 98.33M WeightUpdates/sec
Iter:32/100, MeanErr=1.917581(0.09%), 101.70M WeightUpdates/sec
Iter:33/100, MeanErr=1.913680(-0.20%), 79.94M WeightUpdates/sec
Iter:34/100, MeanErr=1.917264(0.19%), 102.54M WeightUpdates/sec
Iter:35/100, MeanErr=1.917377(0.01%), 100.67M WeightUpdates/sec
Iter:36/100, MeanErr=1.912060(-0.28%), 70.37M WeightUpdates/sec
Iter:37/100, MeanErr=1.917009(0.26%), 80.80M WeightUpdates/sec
Iter:38/100, MeanErr=1.916216(-0.04%), 94.56M WeightUpdates/sec
Iter:39/100, MeanErr=1.916362(0.01%), 28.22M WeightUpdates/sec
Iter:40/100, MeanErr=1.910658(-0.30%), 100.87M WeightUpdates/sec
Iter:41/100, MeanErr=1.916375(0.30%), 85.99M WeightUpdates/sec
Iter:42/100, MeanErr=1.916257(-0.01%), 102.06M WeightUpdates/sec
Iter:43/100, MeanErr=1.914505(-0.09%), 99.86M WeightUpdates/sec
Iter:44/100, MeanErr=1.914638(0.01%), 103.11M WeightUpdates/sec
Iter:45/100, MeanErr=1.915141(0.03%), 107.62M WeightUpdates/sec
Iter:46/100, MeanErr=1.915119(0.00%), 99.65M WeightUpdates/sec
Iter:47/100, MeanErr=1.915379(0.01%), 107.03M WeightUpdates/sec
Iter:48/100, MeanErr=1.912565(-0.15%), 104.78M WeightUpdates/sec
Iter:49/100, MeanErr=1.915466(0.15%), 110.43M WeightUpdates/sec
Iter:50/100, MeanErr=1.914038(-0.07%), 98.44M WeightUpdates/sec
Iter:51/100, MeanErr=1.915015(0.05%), 96.28M WeightUpdates/sec
Iter:52/100, MeanErr=1.913771(-0.06%), 89.27M WeightUpdates/sec
Iter:53/100, MeanErr=1.911621(-0.11%), 72.67M WeightUpdates/sec
Iter:54/100, MeanErr=1.914969(0.18%), 111.17M WeightUpdates/sec
Iter:55/100, MeanErr=1.913894(-0.06%), 98.68M WeightUpdates/sec
Iter:56/100, MeanErr=1.914871(0.05%), 95.41M WeightUpdates/sec
Iter:57/100, MeanErr=1.912898(-0.10%), 80.72M WeightUpdates/sec
Iter:58/100, MeanErr=1.913334(0.02%), 103.71M WeightUpdates/sec
Iter:59/100, MeanErr=1.913362(0.00%), 99.57M WeightUpdates/sec
Iter:60/100, MeanErr=1.913915(0.03%), 106.21M WeightUpdates/sec

Iter:61/100, MeanErr=1.913310(-0.03%), 112.27M WeightUpdates/sec
Iter:62/100, MeanErr=1.913395(0.00%), 50.86M WeightUpdates/sec
Iter:63/100, MeanErr=1.912814(-0.03%), 58.91M WeightUpdates/sec
Iter:64/100, MeanErr=1.911468(-0.07%), 72.06M WeightUpdates/sec
Iter:65/100, MeanErr=1.912313(0.04%), 86.34M WeightUpdates/sec
Iter:66/100, MeanErr=1.913320(0.05%), 114.39M WeightUpdates/sec
Iter:67/100, MeanErr=1.912914(-0.02%), 105.97M WeightUpdates/sec
Iter:68/100, MeanErr=1.909881(-0.16%), 105.73M WeightUpdates/sec
Iter:69/100, MeanErr=1.911649(0.09%), 105.23M WeightUpdates/sec
Iter:70/100, MeanErr=1.911192(-0.02%), 110.24M WeightUpdates/sec
Iter:71/100, MeanErr=1.912480(0.07%), 106.86M WeightUpdates/sec
Iter:72/100, MeanErr=1.909881(-0.14%), 97.28M WeightUpdates/sec
Iter:73/100, MeanErr=1.911678(0.09%), 109.57M WeightUpdates/sec
Iter:74/100, MeanErr=1.911137(-0.03%), 91.01M WeightUpdates/sec
Iter:75/100, MeanErr=1.910706(-0.02%), 99.41M WeightUpdates/sec
Iter:76/100, MeanErr=1.910869(0.01%), 84.18M WeightUpdates/sec
Iter:77/100, MeanErr=1.911643(0.04%), 105.07M WeightUpdates/sec
Iter:78/100, MeanErr=1.911438(-0.01%), 110.12M WeightUpdates/sec
Iter:79/100, MeanErr=1.909590(-0.10%), 84.16M WeightUpdates/sec
Iter:80/100, MeanErr=1.911181(0.08%), 92.30M WeightUpdates/sec
Iter:81/100, MeanErr=1.910534(-0.03%), 110.60M WeightUpdates/sec
Iter:82/100, MeanErr=1.909340(-0.06%), 54.07M WeightUpdates/sec
Iter:83/100, MeanErr=1.908275(-0.06%), 104.08M WeightUpdates/sec
Iter:84/100, MeanErr=1.910364(0.11%), 107.19M WeightUpdates/sec
Iter:85/100, MeanErr=1.910286(0.00%), 102.55M WeightUpdates/sec
Iter:86/100, MeanErr=1.909155(-0.06%), 79.72M WeightUpdates/sec
Iter:87/100, MeanErr=1.909384(0.01%), 102.37M WeightUpdates/sec
Iter:88/100, MeanErr=1.907751(-0.09%), 105.48M WeightUpdates/sec
Iter:89/100, MeanErr=1.910164(0.13%), 102.53M WeightUpdates/sec
Iter:90/100, MeanErr=1.907935(-0.12%), 105.03M WeightUpdates/sec
Iter:91/100, MeanErr=1.909510(0.08%), 99.97M WeightUpdates/sec
Iter:92/100, MeanErr=1.907405(-0.11%), 100.03M WeightUpdates/sec
Iter:93/100, MeanErr=1.905757(-0.09%), 113.21M WeightUpdates/sec
Iter:94/100, MeanErr=1.909167(0.18%), 107.86M WeightUpdates/sec
Iter:95/100, MeanErr=1.907593(-0.08%), 106.09M WeightUpdates/sec
Iter:96/100, MeanErr=1.908358(0.04%), 111.25M WeightUpdates/sec
Iter:97/100, MeanErr=1.906484(-0.10%), 95.81M WeightUpdates/sec
Iter:98/100, MeanErr=1.908239(0.09%), 105.89M WeightUpdates/sec
Iter:99/100, MeanErr=1.908508(0.01%), 103.05M WeightUpdates/sec
Iter:100/100, MeanErr=1.904747(-0.20%), 106.81M WeightUpdates/sec
Done!

Estimated Post-training MeanError = 1.896338

Not training a calibrator because it is not needed.

Elapsed time: 00:00:00.1620840

Elapsed time: 00:00:00.0096627

Beginning processing data.

Rows Read: 38, Read Time: 0, Transform Time: 0

Beginning processing data.

Elapsed time: 00:00:00.0312987

Finished writing 38 rows.

Writing completed.

Rows Read: 5, Total Rows Processed: 5, Total Chunk Time: Less than .001 seconds

Species	Score.0	Score.1	Score.2
---------	---------	---------	---------

```
0 versicolor 0.350161 0.339557 0.310282
1 setosa 0.358506 0.336593 0.304901
2 virginica 0.346957 0.340573 0.312470
3 virginica 0.346685 0.340748 0.312567
4 virginica 0.348469 0.340113 0.311417
```

Regression example

```
...
Regression.
...

import numpy
import pandas
from microsoftml import rx_neural_network, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

attitude = get_dataset("attitude")

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

attituedf = attitude.as_df()
data_train, data_test = train_test_split(attituedf)

model = rx_neural_network(
    formula="rating ~ complaints + privileges + learning + raises + critical
+ advance",
    method="regression",
    data=data_train)

# RuntimeError: The type (RxTextData) for file is not supported.
score_ds = rx_predict(model, data=data_test,
    extra_vars_to_write=["rating"])

# Print the first five rows
print(rx_data_step(score_ds, number_rows_read=5))
```

Output:

```
Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
```

Rows Read: 22, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 22, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Beginning processing data.
Rows Read: 22, Read Time: 0, Transform Time: 0
Beginning processing data.
Using: AVX Math

***** Net definition *****

```
input Data [6];  
hidden H [100] sigmoid { // Depth 1  
    from Data all;  
}  
output Result [1] linear { // Depth 0  
    from H all;  
}
```

***** End net definition *****

Input count: 6
Output count: 1
Output Function: Linear
Loss Function: SquaredLoss
PreTrainer: NoPreTrainer

Starting training...

Learning rate: 0.001000

Momentum: 0.000000

InitWtsDiameter: 0.100000

Initializing 1 Hidden Layers, 801 Weights...

Estimated Pre-training MeanError = 4458.793673

Iter:1/100, MeanErr=1624.747024(-63.56%), 27.30M WeightUpdates/sec
Iter:2/100, MeanErr=139.267390(-91.43%), 30.50M WeightUpdates/sec
Iter:3/100, MeanErr=116.382316(-16.43%), 29.16M WeightUpdates/sec
Iter:4/100, MeanErr=114.947244(-1.23%), 32.06M WeightUpdates/sec
Iter:5/100, MeanErr=112.886818(-1.79%), 32.96M WeightUpdates/sec
Iter:6/100, MeanErr=112.406547(-0.43%), 30.29M WeightUpdates/sec
Iter:7/100, MeanErr=110.502757(-1.69%), 30.92M WeightUpdates/sec
Iter:8/100, MeanErr=111.499645(0.90%), 31.20M WeightUpdates/sec
Iter:9/100, MeanErr=111.895816(0.36%), 32.46M WeightUpdates/sec
Iter:10/100, MeanErr=110.171443(-1.54%), 34.61M WeightUpdates/sec
Iter:11/100, MeanErr=106.975524(-2.90%), 22.14M WeightUpdates/sec
Iter:12/100, MeanErr=107.708220(0.68%), 7.73M WeightUpdates/sec
Iter:13/100, MeanErr=105.345097(-2.19%), 28.99M WeightUpdates/sec
Iter:14/100, MeanErr=109.937833(4.36%), 31.04M WeightUpdates/sec
Iter:15/100, MeanErr=106.672340(-2.97%), 30.04M WeightUpdates/sec
Iter:16/100, MeanErr=108.474555(1.69%), 32.41M WeightUpdates/sec
Iter:17/100, MeanErr=109.449054(0.90%), 31.60M WeightUpdates/sec
Iter:18/100, MeanErr=105.911830(-3.23%), 34.05M WeightUpdates/sec
Iter:19/100, MeanErr=106.045172(0.13%), 33.80M WeightUpdates/sec
Iter:20/100, MeanErr=108.360427(2.18%), 33.60M WeightUpdates/sec
Iter:21/100, MeanErr=106.506436(-1.71%), 33.77M WeightUpdates/sec
Iter:22/100, MeanErr=99.167335(-6.89%), 32.26M WeightUpdates/sec
Iter:23/100, MeanErr=108.115797(9.02%), 25.86M WeightUpdates/sec

Iter:24/100, MeanErr=106.292283(-1.69%), 31.03M WeightUpdates/sec
Iter:25/100, MeanErr=99.397875(-6.49%), 31.33M WeightUpdates/sec
Iter:26/100, MeanErr=104.805299(5.44%), 31.57M WeightUpdates/sec
Iter:27/100, MeanErr=101.385085(-3.26%), 22.92M WeightUpdates/sec
Iter:28/100, MeanErr=100.064656(-1.30%), 35.01M WeightUpdates/sec
Iter:29/100, MeanErr=100.519013(0.45%), 32.74M WeightUpdates/sec
Iter:30/100, MeanErr=99.273143(-1.24%), 35.12M WeightUpdates/sec
Iter:31/100, MeanErr=100.465649(1.20%), 33.68M WeightUpdates/sec
Iter:32/100, MeanErr=102.402320(1.93%), 33.79M WeightUpdates/sec
Iter:33/100, MeanErr=97.517196(-4.77%), 32.32M WeightUpdates/sec
Iter:34/100, MeanErr=102.597511(5.21%), 32.46M WeightUpdates/sec
Iter:35/100, MeanErr=96.187788(-6.25%), 32.32M WeightUpdates/sec
Iter:36/100, MeanErr=101.533507(5.56%), 21.44M WeightUpdates/sec
Iter:37/100, MeanErr=99.339624(-2.16%), 21.53M WeightUpdates/sec
Iter:38/100, MeanErr=98.049306(-1.30%), 15.27M WeightUpdates/sec
Iter:39/100, MeanErr=97.508282(-0.55%), 23.21M WeightUpdates/sec
Iter:40/100, MeanErr=99.894288(2.45%), 27.94M WeightUpdates/sec
Iter:41/100, MeanErr=95.190566(-4.71%), 32.47M WeightUpdates/sec
Iter:42/100, MeanErr=91.234977(-4.16%), 31.29M WeightUpdates/sec
Iter:43/100, MeanErr=98.824414(8.32%), 32.35M WeightUpdates/sec
Iter:44/100, MeanErr=96.759533(-2.09%), 22.37M WeightUpdates/sec
Iter:45/100, MeanErr=95.275106(-1.53%), 32.09M WeightUpdates/sec
Iter:46/100, MeanErr=95.749031(0.50%), 26.49M WeightUpdates/sec
Iter:47/100, MeanErr=96.267879(0.54%), 31.81M WeightUpdates/sec
Iter:48/100, MeanErr=97.383752(1.16%), 31.01M WeightUpdates/sec
Iter:49/100, MeanErr=96.605199(-0.80%), 32.05M WeightUpdates/sec
Iter:50/100, MeanErr=96.927400(0.33%), 32.42M WeightUpdates/sec
Iter:51/100, MeanErr=96.288491(-0.66%), 28.89M WeightUpdates/sec
Iter:52/100, MeanErr=92.751171(-3.67%), 33.68M WeightUpdates/sec
Iter:53/100, MeanErr=88.655001(-4.42%), 34.53M WeightUpdates/sec
Iter:54/100, MeanErr=90.923513(2.56%), 32.00M WeightUpdates/sec
Iter:55/100, MeanErr=91.627261(0.77%), 25.74M WeightUpdates/sec
Iter:56/100, MeanErr=91.132907(-0.54%), 30.00M WeightUpdates/sec
Iter:57/100, MeanErr=95.294092(4.57%), 33.13M WeightUpdates/sec
Iter:58/100, MeanErr=90.219024(-5.33%), 31.70M WeightUpdates/sec
Iter:59/100, MeanErr=92.727605(2.78%), 30.71M WeightUpdates/sec
Iter:60/100, MeanErr=86.910488(-6.27%), 33.07M WeightUpdates/sec
Iter:61/100, MeanErr=92.350984(6.26%), 32.46M WeightUpdates/sec
Iter:62/100, MeanErr=93.208298(0.93%), 31.08M WeightUpdates/sec
Iter:63/100, MeanErr=90.784723(-2.60%), 21.19M WeightUpdates/sec
Iter:64/100, MeanErr=88.685225(-2.31%), 33.17M WeightUpdates/sec
Iter:65/100, MeanErr=91.668555(3.36%), 30.65M WeightUpdates/sec
Iter:66/100, MeanErr=82.607568(-9.88%), 29.72M WeightUpdates/sec
Iter:67/100, MeanErr=88.787842(7.48%), 32.98M WeightUpdates/sec
Iter:68/100, MeanErr=88.793186(0.01%), 34.67M WeightUpdates/sec
Iter:69/100, MeanErr=88.918795(0.14%), 14.09M WeightUpdates/sec
Iter:70/100, MeanErr=87.121434(-2.02%), 33.02M WeightUpdates/sec
Iter:71/100, MeanErr=86.865602(-0.29%), 34.87M WeightUpdates/sec
Iter:72/100, MeanErr=87.261979(0.46%), 32.34M WeightUpdates/sec
Iter:73/100, MeanErr=87.812460(0.63%), 31.35M WeightUpdates/sec
Iter:74/100, MeanErr=87.818462(0.01%), 32.54M WeightUpdates/sec
Iter:75/100, MeanErr=87.085672(-0.83%), 34.80M WeightUpdates/sec
Iter:76/100, MeanErr=85.773668(-1.51%), 35.39M WeightUpdates/sec
Iter:77/100, MeanErr=85.338703(-0.51%), 34.59M WeightUpdates/sec
Iter:78/100, MeanErr=79.370105(-6.99%), 30.14M WeightUpdates/sec

```
Iter:79/100, MeanErr=83.026209(4.61%), 32.32M WeightUpdates/sec
Iter:80/100, MeanErr=89.776417(8.13%), 33.14M WeightUpdates/sec
Iter:81/100, MeanErr=85.447100(-4.82%), 32.32M WeightUpdates/sec
Iter:82/100, MeanErr=83.991969(-1.70%), 22.12M WeightUpdates/sec
Iter:83/100, MeanErr=85.065064(1.28%), 30.41M WeightUpdates/sec
Iter:84/100, MeanErr=83.762008(-1.53%), 31.29M WeightUpdates/sec
Iter:85/100, MeanErr=84.217726(0.54%), 34.92M WeightUpdates/sec
Iter:86/100, MeanErr=82.395181(-2.16%), 34.26M WeightUpdates/sec
Iter:87/100, MeanErr=82.979145(0.71%), 22.87M WeightUpdates/sec
Iter:88/100, MeanErr=83.656685(0.82%), 28.51M WeightUpdates/sec
Iter:89/100, MeanErr=81.132468(-3.02%), 32.43M WeightUpdates/sec
Iter:90/100, MeanErr=81.311106(0.22%), 30.91M WeightUpdates/sec
Iter:91/100, MeanErr=81.953897(0.79%), 31.98M WeightUpdates/sec
Iter:92/100, MeanErr=79.018074(-3.58%), 33.13M WeightUpdates/sec
Iter:93/100, MeanErr=78.220412(-1.01%), 31.47M WeightUpdates/sec
Iter:94/100, MeanErr=80.833884(3.34%), 25.16M WeightUpdates/sec
Iter:95/100, MeanErr=81.550135(0.89%), 32.64M WeightUpdates/sec
Iter:96/100, MeanErr=77.785628(-4.62%), 32.54M WeightUpdates/sec
Iter:97/100, MeanErr=76.438158(-1.73%), 34.34M WeightUpdates/sec
Iter:98/100, MeanErr=79.471621(3.97%), 33.12M WeightUpdates/sec
Iter:99/100, MeanErr=76.038475(-4.32%), 33.01M WeightUpdates/sec
Iter:100/100, MeanErr=75.349164(-0.91%), 32.68M WeightUpdates/sec
Done!
Estimated Post-training MeanError = 75.768932
```

Not training a calibrator because it is not needed.

Elapsed time: 00:00:00.1178557

Elapsed time: 00:00:00.0088299

Beginning processing data.

Rows Read: 8, Read Time: 0, Transform Time: 0

Beginning processing data.

Elapsed time: 00:00:00.0293893

Finished writing 8 rows.

Writing completed.

Rows Read: 5, Total Rows Processed: 5, Total Chunk Time: 0.001 seconds

	rating	Score
0	82.0	70.120613
1	64.0	66.344688
2	68.0	68.862373
3	58.0	68.241341
4	63.0	67.196869

optimizers

- [microsoftml.adadelta_optimizer](#): Adaptive learning rate method
- [microsoftml.sgd_optimizer](#): Stochastic gradient descent

math

- *microsoftml.avx_math*: Acceleration with AVX instructions
- *microsoftml.clr_math*: Acceleration with .NET math
- *microsoftml.gpu_math*: Acceleration with NVidia CUDA
- *microsoftml.mkl_math*: Acceleration with Intel MKL
- *microsoftml.sse_math*: Acceleration with SSE instructions

microsoftml.rx_oneclass_svm: Anomaly Detection

Article • 03/03/2023

Usage

```
microsoftml.rx_oneclass_svm(formula: str,
    data: [revoscalepy.datasources.RxDataSource.RxDataSource,
    pandas.core.frame.DataFrame], cache_size: float = 100,
    kernel: [<function linear_kernel at 0x0000007156EAC8C8>,
    <function polynomial_kernel at 0x0000007156EAC950>,
    <function rbf_kernel at 0x0000007156EAC7B8>,
    <function sigmoid_kernel at 0x0000007156EACA60>] = {'Name': 'RbfKernel',
    'Settings': {}}, epsilon: float = 0.001, nu: float = 0.1,
    shrink: bool = True, normalize: ['No', 'Warn', 'Auto',
    'Yes'] = 'Auto', ml_transforms: list = None,
    ml_transform_vars: list = None, row_selection: str = None,
    transforms: dict = None, transform_objects: dict = None,
    transform_function: str = None,
    transform_variables: list = None,
    transform_packages: list = None,
    transform_environment: dict = None, blocks_per_read: int = None,
    report_progress: int = None, verbose: int = 1,
    ensemble: microsoftml.modules.ensemble.EnsembleControl = None,
    compute_context:
    revoscalepy.computecontext.RxComputeContext.RxComputeContext = None)
```

Description

Machine Learning One Class Support Vector Machines

Details

One-class SVM is an algorithm for anomaly detection. The goal of anomaly detection is to identify outliers that do not belong to some target class. This type of SVM is one-class because the training set contains only examples from the target class. It infers what properties are normal for the objects in the target class and from these properties predicts which examples are unlike the normal examples. This is useful for anomaly detection because the scarcity of training examples is the defining character of

anomalies: typically there are very few examples of network intrusion, fraud, or other types of anomalous behavior.

Arguments

formula

The formula as described in `revoScalePy.rx_formula`. Interaction terms and `F()` are not currently supported in [microsoftml](#).

data

A data source object or a character string specifying a `.xdf` file or a data frame object.

cache_size

The maximal size in MB of the cache that stores the training data. Increase this for large training sets. The default value is 100 MB.

kernel

A character string representing the kernel used for computing inner products. For more information, see `ma_kernel()`. The following choices are available:

- `rbf_kernel`: Radial basis function kernel. Its parameter represents `gamma` in the term $\exp(-\text{gamma}|x-y|^2)$. If not specified, it defaults to `1` divided by the number of features used. For example, `rbf_kernel(gamma = .1)`. This is the default value.
- `linear_kernel`: Linear kernel.
- `polynomial_kernel`: Polynomial kernel with parameter names `a`, `bias`, and `deg` in the term $(a \cdot \langle x, y \rangle + \text{bias})^{\text{deg}}$. The `bias`, defaults to `0`. The degree, `deg`, defaults to `3`. If `a` is not specified, it is set to `1` divided by the number of features.
- `sigmoid_kernel`: Sigmoid kernel with parameter names `gamma` and `coef0` in the term $\tanh(\text{gamma} \cdot \langle x, y \rangle + \text{coef0})$. `gamma`, defaults to `1` divided by the number of features. The parameter `coef0` defaults to `0`. For example, `sigmoid_kernel(gamma = .1, coef0 = 0)`.

epsilon

The threshold for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model. The value must be greater than or equal to `numpy.finfo(double).eps`. The default value is 0.001.

nu

The trade-off between the fraction of outliers and the number of support vectors (represented by the Greek letter nu). Must be between 0 and 1, typically between 0.1 and 0.5. The default value is 0.1.

shrink

Uses the shrinking heuristic if `True`. In this case, some samples will be "shrunk" during the training procedure, which may speed up training. The default value is `True`.

normalize

Specifies the type of automatic normalization used:

- `"Auto"`: if normalization is needed, it is performed automatically. This is the default choice.
- `"No"`: no normalization is performed.
- `"Yes"`: normalization is performed.
- `"Warn"`: if normalization is needed, a warning message is displayed, but normalization is not performed.

Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MinMax` normalizer is used. It normalizes values in an interval $[a, b]$ where $-1 \leq a \leq 0$ and $0 \leq b \leq 1$ and $b - a = 1$. This normalizer preserves sparsity by mapping zero to zero.

ml_transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or *None* if no transforms are to be performed. See [featurize_text](#), [categorical](#), and

`categorical_hash`, for transformations that are supported. These transformations are performed after any specified Python transformations. The default value is *None*.

`ml_transform_vars`

Specifies a character vector of variable names to be used in `ml_transforms` or *None* if none are to be used. The default value is *None*.

`row_selection`

NOT SUPPORTED. Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example:

- `row_selection = "old"` will only use observations in which the value of the variable `old` is `True`.
- `row_selection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10.

The row selection is performed after processing any data transformations (see the arguments `transforms` or `transform_function`). As with all expressions, `row_selection` can be defined outside of the function call using the `expression` function.

`transforms`

NOT SUPPORTED. An expression of the form that represents the first round of variable transformations. As with all expressions, `transforms` (or `row_selection`) can be defined outside of the function call using the `expression` function.

`transform_objects`

NOT SUPPORTED. A named list that contains objects that can be referenced by `transforms`, `transform_function`, and `row_selection`.

`transform_function`

The variable transformation function.

transform_variables

A character vector of input data set variables needed for the transformation function.

transform_packages

NOT SUPPORTED. A character vector specifying additional Python packages (outside of those specified in `RxOptions.get_option("transform_packages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in `revoscalepy` functions via their `transforms` and `transform_function` arguments or those defined implicitly via their `formula` or `row_selection` arguments. The `transform_packages` argument may also be `None`, indicating that no packages outside `RxOptions.get_option("transform_packages")` are preloaded.

transform_environment

NOT SUPPORTED. A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transform_environment = None`, a new "hash" environment with parent `revoscalepy.baseenv` is used instead.

blocks_per_read

Specifies the number of blocks to read for each chunk of data read from the data source.

report_progress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of

information.

compute_context

Sets the context in which computations are executed, specified with a valid `revoscalepy.RxComputeContext`. Currently local and [revoscalepy.RxInSqlServer](#) compute contexts are supported.

ensemble

Control parameters for ensembling.

Returns

A [OneClassSvm](#) object with the trained model.

Note

This algorithm is single-threaded and will always attempt to load the entire dataset into memory.

See also

[linear_kernel](#), [polynomial_kernel](#), [rbf_kernel](#), [sigmoid_kernel](#), [rx_predict](#).

References

[Wikipedia: Anomaly detection](#) [↗](#)

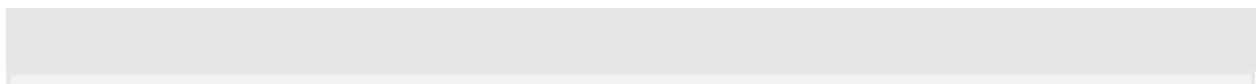
[Microsoft Azure Machine Learning Studio \(classic\): One-Class Support Vector Machine](#)

[Estimating the Support of a High-Dimensional Distribution](#) [↗](#)

[New Support Vector Algorithms](#) [↗](#)

[LIBSVM: A Library for Support Vector Machines](#) [↗](#)

Example



```

...
Anomaly Detection.
...
import numpy
import pandas
from microsoftml import rx_oneclass_svm, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

iris = get_dataset("iris")

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

irisdf = iris.as_df()
data_train, data_test = train_test_split(irisdf)

# Estimate a One-Class SVM model
model = rx_oneclass_svm(
    formula= "~ Sepal_Length + Sepal_Width + Petal_Length +
Petal_Width",
    data=data_train)

# Add additional non-iris data to the test data set
data_test["isIris"] = 1.0
not_iris = pandas.DataFrame(data=dict(Sepal_Length=[2.5, 2.6],
    Sepal_Width=[.75, .9], Petal_Length=[2.5, 2.5],
    Petal_Width=[.8, .7], Species=["not iris", "not iris"],
    isIris=[0., 0.]))

merged_test = pandas.concat([data_test, not_iris])

scoresdf = rx_predict(model, data=merged_test, extra_vars_to_write=
["isIris"])

# Look at the last few observations
print(scoresdf.tail())

```

Output:

```

Automatically adding a MinMax normalization transform, use 'norm=Warn' or
'norm=No' to turn this behavior off.
Beginning processing data.
Rows Read: 112, Read Time: 0, Transform Time: 0
Beginning processing data.
Beginning processing data.
Beginning processing data.
Rows Read: 112, Read Time: 0, Transform Time: 0
Beginning processing data.

```

Using these libsvm parameters: svm_type=2, nu=0.1, cache_size=100,
eps=0.001, shrinking=1, kernel_type=2, gamma=0.25, degree=0, coef0=0

Reconstructed gradient.

optimization finished, #iter = 15

obj = 52.905421, rho = 9.506052

nSV = 12, nBSV = 9

Not training a calibrator because it is not needed.

Elapsed time: 00:00:00.0555122

Elapsed time: 00:00:00.0212389

Beginning processing data.

Rows Read: 40, Read Time: 0, Transform Time: 0

Beginning processing data.

Elapsed time: 00:00:00.0349974

Finished writing 40 rows.

Writing completed.

	isIris	Score
35	1.0	-0.142141
36	1.0	-0.531449
37	1.0	-0.189874
38	0.0	0.635845
39	0.0	0.555602

microsoftml.rx_predict: Scores using a Microsoft machine learning model

Article • 03/03/2023

Usage

```
microsoftml.rx_predict(model,
    data: typing.Union[revoscalepy.datasources.RxDataSource.RxDataSource,
    pandas.core.frame.DataFrame],
    output_data:
typing.Union[revoscalepy.datasources.RxDataSource.RxDataSource,
    str] = None, write_model_vars: bool = False,
    extra_vars_to_write: list = None, suffix: str = None,
    overwrite: bool = False, data_threads: int = None,
    blocks_per_read: int = None, report_progress: int = None,
    verbose: int = 1,
    compute_context:
revoscalepy.computecontext.RxComputeContext.RxComputeContext = None,
    **kargs)
```

Description

Reports per-instance scoring results in a data frame or revoscalepy data source using a trained Microsoft ML Machine Learning model with arevoscalepydata source.

Details

The following items are reported in the output by default: scoring on three variables for the binary classifiers: PredictedLabel, Score, and Probability; the Score for oneClassSvm and regression classifiers; PredictedLabel for Multi-class classifiers, plus a variable for each category prepended by the Score.

Arguments

model

A model information object returned from a microsoftml model. For example, an object returned from `rx_fast_trees` or `rx_logistic_regression`.

data

A [revoscalepy](#) data source object, a data frame, or the path to a `.xdf` file.

output_data

Output text or xdf file name or an `RxDataSource` with write capabilities in which to store transformed data. If `None`, a data frame is returned. The default value is `None`.

write_model_vars

If `True`, variables in the model are written to the output data set in addition to the scoring variables. If variables from the input data set are transformed in the model, the transformed variables are also included. The default value is `False`.

extra_vars_to_write

`None` or character vector of additional variables names from the input data to include in the `output_data`. If `write_model_vars` is `True`, model variables are included as well. The default value is `None`.

suffix

A character string specifying suffix to append to the created scoring variable(s) or `None` if there is no suffix. The default value is `None`.

overwrite

If `True`, an existing `output_data` is overwritten; if `False` an existing `output_data` is not overwritten. The default value is `False`.

data_threads

An integer specifying the desired degree of parallelism in the data pipeline. If `None`, the number of threads used is determined internally. The default value is `None`.

blocks_per_read

Specifies the number of blocks to read for each chunk of data read from the data source.

report_progress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

The default value is `1`.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information. The default value is `1`.

compute_context

Sets the context in which computations are executed, specified with a valid `revoscalepy.RxComputeContext`. Currently `local` and `revoscalepy.RxInSqlServer` compute contexts are supported.

kargs

Additional arguments sent to compute engine.

Returns

A data frame or an `revoscalepy.RxDataSource` object representing the created output data. By default, output from scoring binary classifiers include three variables:

`PredictedLabel`, `Score`, and `Probability`; `rx_oneclass_svm` and regression include one variable: `Score`; and multi-class classifiers include `PredictedLabel` plus a variable for

each category prepended by `Score`. If a `suffix` is provided, it is added to the end of these output variable names.

See also

[rx_featurize](#), [revoscalepy.rx_data_step](#), [revoscalepy.rx_import](#).

Binary classification example

```
...
Binary Classification.
...

import numpy
import pandas
from microsoftml import rx_fast_linear, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

infert = get_dataset("infert")

import sklearn
if sklearn.__version__ < "0.18":
    from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

infertdf = infert.as_df()
infertdf["isCase"] = infertdf.case == 1
data_train, data_test, y_train, y_test = train_test_split(infertdf,
infertdf.isCase)

forest_model = rx_fast_linear(
    formula=" isCase ~ age + parity + education + spontaneous + induced ",
    data=data_train)

# RuntimeError: The type (RxTextData) for file is not supported.
score_ds = rx_predict(forest_model, data=data_test,
                      extra_vars_to_write=["isCase", "Score"])

# Print the first five rows
print(rx_data_step(score_ds, number_rows_read=5))
```

Output:

Automatically adding a MinMax normalization transform, use 'norm=Warn' or 'norm=No' to turn this behavior off.

Beginning processing data.

Rows Read: 186, Read Time: 0, Transform Time: 0

Beginning processing data.

Beginning processing data.

Rows Read: 186, Read Time: 0.001, Transform Time: 0

Beginning processing data.

Beginning processing data.

Rows Read: 186, Read Time: 0.001, Transform Time: 0

Beginning processing data.

Using 2 threads to train.

Automatically choosing a check frequency of 2.

Auto-tuning parameters: maxIterations = 8064.

Auto-tuning parameters: L2 = 2.666837E-05.

Auto-tuning parameters: L1Threshold (L1/L2) = 0.

Using best model from iteration 590.

Not training a calibrator because it is not needed.

Elapsed time: 00:00:00.6058289

Elapsed time: 00:00:00.0084728

Beginning processing data.

Rows Read: 62, Read Time: 0, Transform Time: 0

Beginning processing data.

Elapsed time: 00:00:00.0302359

Finished writing 62 rows.

Writing completed.

Rows Read: 5, Total Rows Processed: 5, Total Chunk Time: 0.001 seconds

	isCase	PredictedLabel	Score	Probability
0	False	True	0.576775	0.640325
1	False	False	-2.929549	0.050712
2	True	False	-2.370090	0.085482
3	False	False	-1.700105	0.154452
4	False	False	-0.110981	0.472283

Regression example

```
...
Regression.
...

import numpy
import pandas
from microsoftml import rx_fast_trees, rx_predict
from revoscalepy.etl.RxDataStep import rx_data_step
from microsoftml.datasets.datasets import get_dataset

airquality = get_dataset("airquality")

import sklearn
if sklearn.__version__ < "0.18":
```



```

from sklearn.cross_validation import train_test_split
else:
    from sklearn.model_selection import train_test_split

airquality = airquality.as_df()

#####
# Estimate a regression fast forest
# Use the built-in data set 'airquality' to create test and train data

df = airquality[airquality.Ozone.notnull()]
df["Ozone"] = df.Ozone.astype(float)

data_train, data_test, y_train, y_test = train_test_split(df, df.Ozone)

airFormula = " Ozone ~ Solar_R + Wind + Temp "

# Regression Fast Forest for train data
ff_reg = rx_fast_trees(airFormula, method="regression", data=data_train)

# Put score and model variables in data frame
score_df = rx_predict(ff_reg, data=data_test, write_model_vars=True)
print(score_df.head())

# Plot actual versus predicted values with smoothed line
# Supported in the next version.
# rx_line_plot(" Score ~ Ozone ", type=["p", "smooth"], data=score_df)

```

Output:

```

'unbalanced_sets' ignored for method 'regression'
Not adding a normalizer.
Making per-feature arrays
Changing data from row-wise to column-wise
Beginning processing data.
Rows Read: 87, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Warning: Skipped 4 instances with missing features during training
Processed 83 instances
Binning and forming Feature objects
Reserved memory for tree learner: 22620 bytes
Starting to train ...
Not training a calibrator because it is not needed.
Elapsed time: 00:00:00.0390764
Elapsed time: 00:00:00.0080750
Beginning processing data.
Rows Read: 29, Read Time: 0.001, Transform Time: 0
Beginning processing data.
Elapsed time: 00:00:00.0221875
Finished writing 29 rows.

```

Writing completed.

	Solar_R	Wind	Temp	Score
0	290.0	9.2	66.0	33.195541
1	259.0	15.5	77.0	20.906796
2	276.0	5.1	88.0	76.594643
3	139.0	10.3	81.0	31.668842
4	236.0	14.9	81.0	43.590839

microsoftml.select_columns: Retains columns of a dataset

Article • 03/03/2023

Usage

```
microsoftml.select_columns(cols: [list, str], **kargs)
```

Description

Selects a set of columns to retrain, dropping all others.

Arguments

cols

A character string or list of the names of the variables to keep.

kargs

Additional arguments sent to compute engine.

Returns

An object defining the transform.

See also

[concat](#), [drop_columns](#).

microsoftml.sgd_optimizer: Stochastic gradient descent

Article • 03/03/2023

Usage

```
microsoftml.sgd_optimizer(learning_rate: numbers.Real = None,  
    momentum: numbers.Real = None, nag: bool = None,  
    weight_decay: numbers.Real = None,  
    l_rate_red_ratio: numbers.Real = None,  
    l_rate_red_freq: numbers.Real = None,  
    l_rate_red_error_ratio: numbers.Real = None)
```

Description

Stochastic gradient descent optimizer.

Arguments

learning_rate

Learning rate (settings).

momentum

Momentum Term (settings).

nag

Use Nesterov's accelerated gradient (settings).

weight_decay

Weight decay (settings).

l_rate_red_ratio

Learning rate reduction ratio (settings).

l_rate_red_freq

Learning rate reduction ratio (settings).

l_rate_red_error_ratio

Relative error reduction criterion for learning rate reduction (settings).

See also

[adadelta_optimizer](#)

microsoftml.smoothed_hinge_loss: Smoothed hinge loss function

Article • 03/03/2023

Usage

```
microsoftml.smoothed_hinge_loss(smoothing_const: numbers.Real = 1.0)
```

Description

Smoothed Hinge loss.

Arguments

smoothing_const

Smoothing constant (settings).

See also

[hinge_loss](#), [log_loss](#), [squared_loss](#)

microsoftml.squared_loss: Squared loss function

Article • 03/03/2023

Usage

```
microsoftml.squared_loss()
```

Description

Squared loss.

See also

[hinge_loss](#), [log_loss](#), [smoothed_hinge_loss](#)

microsoftml.sse_math: Acceleration with SSE instructions

Article • 03/03/2023

Usage

```
microsoftml.sse_math()
```

Description

Implementation accelerated with SSE instructions.

See also

[avx_math](#), [clr_math](#), [gpu_math](#), [mkl_math](#)

revoscalepy (Python package in SQL Server Machine Learning Services)

Article • 02/28/2023

Applies to:  SQL Server 2017 (14.x) and later

revoscalepy is a Python package from Microsoft that supports distributed computing, remote compute contexts, and high-performance data science algorithms. The package is included in [SQL Server Machine Learning Services](#).

The package offers the following functionality:

- Local and remote compute contexts on systems having the same version of **revoscalepy**
- Data transformation and visualization functions
- Data science functions, scalable through distributed or parallel processing
- Improved performance, including use of the Intel math libraries

Data sources and compute contexts that you create in **revoscalepy** can also be used in machine learning algorithms. For an introduction to these algorithms, see [microsoftml Python module in SQL Server](#).


Full reference documentation

The **revoscalepy** package is distributed in multiple Microsoft products, but usage is the same whether you get the package in SQL Server or another product. Because the functions are the same, [documentation for individual revoscalepy functions](#) is published to just one location under the [Python reference](#). Should any product-specific behaviors exist, discrepancies will be noted in the function help page.

Versions and platforms

The **revoscalepy** module is based on Python 3.5 and available only when you install one of the following Microsoft products or downloads:

- [SQL Server Machine Learning Services](#)
- [Python client libraries for a data science client](#)

 **Note**

Full product release versions are Windows-only in SQL Server 2017. Both Windows and Linux are supported for **revoscalepy** in SQL Server 2019 and later.

Functions by category

This section lists the functions by category to give you an idea of how each one is used. You can also use the [table of contents](#) to find functions in alphabetical order.

1-Data source and compute

revoscalepy includes functions for creating data sources and setting the location, or *compute context*, of where computations are performed. Functions relevant to SQL Server scenarios are listed in the table below.

SQL Server and Python use different data types in some cases. For a list of mappings between SQL and Python data types, see [Python-to-SQL data types](#).

Function	Description
RxInSqlServer	Create a SQL Server compute context object to push computations to a remote instance. Several revoscalepy functions take compute context as an argument. For a context-switch example, see Create a model using revoscalepy .
RxSqlServerData	Create a data object based on a SQL Server query or table.
RxOdbcData	Create a data source based on an ODBC connection.
RxxdfData	Create a data source based on a local XDF file. XDF files are often used to offload in-memory data to disk. An XDF file can be useful when working with more data than can be transferred from the database in one batch, or more data than can fit in memory. For example, if you regularly move large amounts of data from a database to a local workstation, rather than query the database repeatedly for each R operation, you can use the XDF file as a kind of cache to save the data locally and then work with it in your R workspace.

Tip

If you are new to the idea of data sources or compute contexts, we recommend that you start with the article [Distributed computing](#).

2-Data manipulation (ETL)

Function	Description
rx_import	Import data into a .xdf file or data frame.
rx_data_step	Transform data from an input data set to an output data set.

3-Training and summarization

Function	Description
rx_btrees	Fit stochastic gradient boosted decision trees
rx_dforest	Fit classification and regression decision forests
rx_dtree	Fit classification and regression trees
rx_lin_mod	Create a linear regression model
rx_logit	Create a logistic regression model
rx_summary	Produce univariate summaries of objects in revoscalepy.

You should also review the functions in [microsoftml](#) for additional approaches.

4-Scoring functions

Function	Description
rx_predict	Generate predictions from a trained model and can be used for real-time scoring.
rx_predict_default	Compute predicted values and residuals using rx_lin_mod and rx_logit objects.
rx_predict_rx_dforest	Calculate predicted or fitted values for a data set from an rx_dforest or rx_btrees object.
rx_predict_rx_dtree	Calculate predicted or fitted values for a data set from an rx_dtree object.

How to work with revoscalepy

Functions in **revoscalepy** are callable in Python code encapsulated in stored procedures. Most developers build **revoscalepy** solutions locally, and then migrate finished Python code to stored procedures as a deployment exercise.

When running locally, you typically run a Python script from the command line, or from a Python development environment, and specify a SQL Server compute context using one of the **revoscalepy** functions. You can use the remote compute context for the entire code, or for individual functions. For example, you might want to offload model training to the server to use the latest data and avoid data movement.

When you are ready to encapsulate Python script inside a stored procedure, [sp_execute_external_script](#), we recommend rewriting the code as a single function that has clearly defined inputs and outputs.

Inputs and outputs must be **pandas** data frames. When this is done, you can call the stored procedure from any client that supports T-SQL, easily pass SQL queries as inputs, and save the results to SQL tables. For an example, see [Learn in-database Python analytics for SQL developers](#).

Using revoscalepy with microsoftml

The Python functions for [microsoftml](#) are integrated with the compute contexts and data sources that are provided in revoscalepy. When calling functions from microsoftml, for example when defining and training a model, use the revoscalepy functions to execute the Python code either locally or in a SQL Server remote compute context.

The following example shows the syntax for importing modules in your Python code. You can then reference the individual functions you need.

Python

```
from microsoftml.modules.logistic_regression.rx_logistic_regression import  
rx_logistic_regression  
from revoscalepy.functions.RxSummary import rx_summary  
from revoscalepy.etl.RxImport import rx_import_datasource
```

See also

- [Python tutorials](#)
- [Python Reference](#)

RevoScaleR (R package in SQL Server Machine Learning Services)

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x) and later versions

RevoScaleR is an R package from Microsoft that supports distributed computing, remote compute contexts, and high-performance data science algorithms. It also supports data import, data transformation, summarization, visualization, and analysis. The package is included in [SQL Server Machine Learning Services](#) and [SQL Server 2016 R Services](#).

In contrast with base R functions, RevoScaleR operations can be performed against large datasets, in parallel, and on distributed file systems. Functions can operate over datasets that do not fit in memory by using chunking and by reassembling results when operations are complete.

RevoScaleR functions are denoted with a `rx**` or `Rx` prefix to make them easy to identify.

RevoScaleR serves as a platform for distributed data science. For example, you can use the RevoScaleR compute contexts and transformations with the state-of-the-art algorithms in [MicrosoftML](#). You can also use [rxExec](#) to run base R functions in parallel.

Full reference documentation

The **RevoScaleR** package is distributed in multiple Microsoft products, but usage is the same whether you get the package in SQL Server or another product. Because the functions are the same, [documentation for individual RevoScaleR functions](#) is published to just one location under the [R reference](#). Should any product-specific behaviors exist, discrepancies will be noted in the function help page.

Versions and platforms

The **RevoScaleR** package is based on R 3.4.3 and available only when you install one of the following Microsoft products or downloads:

- [SQL Server 2016 R Services](#)
- [SQL Server Machine Learning Services](#)
- [Microsoft R client](#)

ⓘ Note

Full product release versions are Windows-only in SQL Server 2017. Both Windows and Linux are supported for **RevoScaleR** in **SQL Server 2019**.

Functions by category

This section lists the functions by category to give you an idea of how each one is used. You can also use the [table of contents](#) to find functions in alphabetical order.

1-Data source and compute

RevoScaleR includes functions for creating data sources and setting the location, or *compute context*, of where computations are performed. A data source object is a container that specifies a connection string together with the set of data that you want, defined either as a table, view, or query. Stored procedure calls are not supported. Functions relevant to SQL Server scenarios are listed in the table below.

SQL Server and R use different data types in some cases. For a list of mappings between SQL and R data types, see [R-to-SQL data types](#).

Function	Description
RxInSqlServer	Create a SQL Server compute context object to push computations to a remote instance. Several RevoScaleR functions take compute context as an argument.
rxGetComputeContext / rxSetComputeContext	Get or set the active compute context.
RxSqlServerData	Create a data object based on a SQL Server query or table.
RxOdbcData	Create a data source based on an ODBC connection.
RxXdfData	Create a data source based on a local XDF file. XDF files are often used to offload in-memory data to disk. An XDF file can be useful when working with more data than can be transferred from the database in one batch, or more data than can fit in memory. For example, if you regularly move large amounts of data from a database to a local workstation, rather than query the database repeatedly for each R operation, you can use the XDF file as a kind of cache to save the data locally and then work with it in your R workspace.

💡 Tip

If you are new to the idea of data sources or compute contexts, we recommend that you start with the article [Distributed computing](#).

Perform DDL statements

You can execute DDL statements from R, if you have the necessary permissions on the instance and database. The following functions use ODBC calls to execute DDL statements or retrieve the database schema.

Function	Description
rxSqlServerTableExists and rxSqlServerDropTable	Drop a SQL Server table, or check for the existence of a database table or object.
rxExecuteSQLDDL	Execute a Data Definition Language (DDL) command that defines or manipulates database objects. This function cannot return data, and is used only to retrieve or modify the object schema or metadata.

2-Data manipulation (ETL)

After you have created a data source object, you can use the object to load data into it, transform data, or write new data to the specified destination. Depending on the size of the data in the source, you can also define the batch size as part of the data source and move data in chunks.

Function	Description
rxOpen-methods	Check whether a data source is available, open or close a data source, read data from a source, write data to the target, and close a data source.
rxImport	Move data from a data source into file storage or into a data frame.
rxDataStep	Transform data while moving it between data sources.

3-Graphing functions

Function name	Description
rxHistogram	Creates a histogram from data.

Function name	Description
rxLinePlot	Creates a line plot from data.
rxLorenz	Computes a Lorenz curve, which can be plotted.
rxRocCurve	Computes and plots ROC curves from actual and predicted data.

4-Descriptive statistics

Function name	Description
rxQuantile *	Computes approximate quantiles for .xdf files and data frames without sorting.
rxSummary *	Basic summary statistics of data, including computations by group. Writing by group computations to .xdf file not supported.
rxCrossTabs *	Formula-based cross-tabulation of data.
rxCube *	Alternative formula-based cross-tabulation designed for efficient representation returning cube results. Writing output to .xdf file not supported.
rxMarginals	Marginal summaries of cross-tabulations.
as.xtabs	Converts cross tabulation results to an xtabs object.
rxChiSquaredTest	Performs Chi-squared Test on xtabs object. Used with small data sets and does not chunk data.
rxFisherTest	Performs Fisher's Exact Test on xtabs object. Used with small data sets and does not chunk data.
rxKendallCor	Computes Kendall's Tau Rank Correlation Coefficient using xtabs object.
rxPairwiseCrossTab	Apply a function to pairwise combinations of rows and columns of an xtabs object.
rxRiskRatio	Calculate the relative risk on a two-by-two xtabs object.
rxOddsRatio	Calculate the odds ratio on a two-by-two xtabs object.

* Signifies the most popular functions in this category.

5-Prediction functions

Function name	Description
rxLinMod *	Fits a linear model to data.
rxLogit *	Fits a logistic regression model to data.
rxGlm *	Fits a generalized linear model to data.
rxCovCor *	Calculate the covariance, correlation, or sum of squares / cross-product matrix for a set of variables.
rxDTree *	Fits a classification or regression tree to data.
rxBTrees *	Fits a classification or regression decision forest to data using a stochastic gradient boosting algorithm.
rxDForest *	Fits a classification or regression decision forest to data.
rxPredict *	Calculates predictions for fitted models. Output must be an XDF data source.
rxKmeans *	Performs k-means clustering.
rxNaiveBayes *	Performs Naive Bayes classification.
rxCov	Calculate the covariance matrix for a set of variables.
rxCor	Calculate the correlation matrix for a set of variables.
rxSSCP	Calculate the sum of squares / cross-product matrix for a set of variables.
rxRoc	Receiver Operating Characteristic (ROC) computations using actual and predicted values from binary classifier system.

* Signifies the most popular functions in this category.

How to work with RevoScaleR

Functions in **RevoScaleR** are callable in R code encapsulated in stored procedures. Most developers build **RevoScaleR** solutions locally, and then migrate finished R code to stored procedures as a deployment exercise.

When running locally, you typically run an R script from the command line, or from an R development environment, and specify a SQL Server compute context using one of the **RevoScaleR** functions. You can use the remote compute context for the entire code, or for individual functions. For example, you might want to offload model training to the server to use the latest data and avoid data movement.

When you are ready to encapsulate R script inside a stored procedure, [sp_execute_external_script](#), we recommend rewriting the code as a single function that has clearly defined inputs and outputs.

See also

- [R tutorials](#)
- [Learn to use compute contexts](#)
- [R for SQL developers: Train and operationalize a model](#)
- [Microsoft product samples on GitHub](#) [↗](#)
- [R reference](#)

MicrosoftML (R package in SQL Server Machine Learning Services)

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x) and later versions

MicrosoftML is an R package from Microsoft that provides high-performance machine learning algorithms. It includes functions for training and transformations, scoring, text and image analysis, and feature extraction for deriving values from existing data. The package is included in [SQL Server Machine Learning Services](#) and [SQL Server 2016 R Services](#) and supports high performance on big data, using multicore processing, and fast data streaming. MicrosoftML also includes numerous transformations for text and image processing.

Full reference documentation

The **MicrosoftML** package is distributed in multiple Microsoft products, but usage is the same whether you get the package in SQL Server or another product. Because the functions are the same, [documentation for individual RevoScaleR functions](#) is published to just one location under the [R reference](#). Should any product-specific behaviors exist, discrepancies will be noted in the function help page.

Versions and platforms

The **MicrosoftML** package is based on R 3.5.2 and available only when you install one of the following Microsoft products or downloads:

- [SQL Server 2016 R Services](#)
- [SQL Server Machine Learning Services](#)
- [Microsoft R client](#)

Note

Full product release versions are Windows-only in SQL Server 2017. Both Windows and Linux are supported for **MicrosoftML** in [SQL Server 2019](#).

Package dependencies

Algorithms in **MicrosoftML** depend on [RevoScaleR](#) for:

- Data source objects. Data consumed by **MicrosoftML** functions are created using **RevoScaleR** functions.
- Remote computing (shifting function execution to a remote SQL Server instance). The **RevoScaleR** package provides functions for creating and activating a remote compute context for SQL server.

In most cases, you will load the packages together whenever you are using **MicrosoftML**.

Functions by category

This section lists the functions by category to give you an idea of how each one is used. You can also use the [table of contents](#) to find functions in alphabetical order.

1-Machine learning algorithms

Function name	Description
rxFastTrees	An implementation of FastRank, an efficient implementation of the MART gradient boosting algorithm.
rxFastForest	A random forest and Quantile regression forest implementation using rxFastTrees .
rxLogisticRegression	Logistic regression using L-BFGS.
rxOneClassSvm	One class support vector machines.
rxNeuralNet	Binary, multi-class, and regression neural net.
rxFastLinear	Stochastic dual coordinate ascent optimization for linear binary classification and regression.
rxEnsemble	Trains a number of models of various kinds to obtain better predictive performance than could be obtained from a single model.

2-Transformation functions

Function name	Description
concat	Transformation to create a single vector-valued column from multiple columns.
categorical	Create indicator vector using categorical transform with dictionary.

Function name	Description
categoricalHash	Converts the categorical value into an indicator array by hashing.
featurizeText	Produces a bag of counts of sequences of consecutive words, called n-grams, from a given corpus of text. It offers language detection, tokenization, stopwords removing, text normalization, and feature generation.
getSentiment	Scores natural language text and creates a column that contains probabilities that the sentiments in the text are positive.
ngram	allows defining arguments for count-based and hash-based feature extraction.
selectColumns	Selects a set of columns to retrain, dropping all others.
selectFeatures	Selects features from the specified variables using a specified mode.
loadImage	Loads image data.
resizeImage	Resizes an image to a specified dimension using a specified resizing method.
extractPixels	Extracts the pixel values from an image.
featurizeImage	Featurizes an image using a pre-trained deep neural network model.

3-Scoring and training functions

Function name	Description
rxPredict.mlModel	Runs the scoring library either from SQL Server, using the stored procedure, or from R code enabling real-time scoring to provide much faster prediction performance.
rxFeaturize	Transforms data from an input data set to an output data set.
mlModel	Provides a summary of a Microsoft R Machine Learning model.

4-Loss functions for classification and regression

Function name	Description
expLoss	Specifications for exponential classification loss function.
logLoss	Specifications for log classification loss function.
hingeLoss	Specifications for hinge classification loss function.

Function name	Description
smoothHingeLoss	Specifications for smooth hinge classification loss function.
poissonLoss	Specifications for poisson regression loss function.
squaredLoss	Specifications for squared regression loss function.

5-Feature selection functions

Function name	Description
minCount	Specification for feature selection in count mode.
mutualInformation	Specification for feature selection in mutual information mode.

6-Ensemble modeling functions

Function name	Description
fastTrees	Creates a list containing the function name and arguments to train a Fast Tree model with rxEnsemble .
fastForest	Creates a list containing the function name and arguments to train a Fast Forest model with rxEnsemble .
fastLinear	Creates a list containing the function name and arguments to train a Fast Linear model with rxEnsemble .
logisticRegression	Creates a list containing the function name and arguments to train a Logistic Regression model with rxEnsemble .
oneClassSvm	Creates a list containing the function name and arguments to train a OneClassSvm model with rxEnsemble .

7-Neural networking functions

Function name	Description
optimizer	Specifies optimization algorithms for the rxNeuralNet machine learning algorithm.

8-Package state functions

Function name	Description
rxHashEnv	An environment object used to store package-wide state.

How to use MicrosoftML

Functions in **MicrosoftML** are callable in R code encapsulated in stored procedures. Most developers build **MicrosoftML** solutions locally, and then migrate finished R code to stored procedures as a deployment exercise.

The **MicrosoftML** package for R is installed "out-of-the-box" in SQL Server 2017.

The package is not loaded by default. As a first step, load the **MicrosoftML** package, and then load **RevoScaleR** if you need to use remote compute contexts or related connectivity or data source objects. Then, reference the individual functions you need.

```
R
```

```
library(microsoftml);  
library(RevoScaleR);  
logisticRegression(args);
```

See also

- [R tutorials](#)
- [Learn to use compute contexts](#)
- [R for SQL developers: Train and operationalize a model](#)
- [Microsoft product samples on GitHub](#) ↗
- [R reference](#)

categorical: Machine Learning Categorical Data Transform

Article • 02/28/2023

Categorical transform that can be performed on data before training a model.

Usage

```
categorical(vars, outputKind = "ind", maxNumTerms = 1e+06, terms = "", ...)
```

Arguments

vars

A character vector or list of variable names to transform. If named, the names represent the names of new variables to be created.

outputKind

A character string that specifies the kind of output kind.

- **"ind"**: Outputs an indicator vector. The input column is a vector of categories, and the output contains one indicator vector per slot in the input column.
- **"bag"**: Outputs a multi-set vector. If the input column is a vector of categories, the output contains one vector, where the value in each slot is the number of occurrences of the category in the input vector. If the input column contains a single category, the indicator vector and the bag vector are equivalent
- **"key"**: Outputs an index. The output is an integer ID (between 1 and the number of categories in the dictionary) of the category.

The default value is **"ind"**.

maxNumTerms

An integer that specifies the maximum number of categories to include in the dictionary. The default value is 1000000.

terms

Optional character vector of terms or categories.



Additional arguments sent to compute engine.

Details

The `categorical` transform passes through a data set, operating on text columns, to build a dictionary of categories. For each row, the entire text string appearing in the input column is defined as a category. The output of the categorical transform is an indicator vector. Each slot in this vector corresponds to a category in the dictionary, so its length is the size of the built dictionary. The categorical transform can be applied to one or more columns, in which case it builds a separate dictionary for each column that it is applied to.

`categorical` is not currently supported to handle factor data.

Value

A `map1` object defining the transform.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

See also

[rxFastTrees](#), [rxFastForest](#), [rxNeuralNet](#), [rxOneClassSvm](#), [rxLogisticRegression](#).

Examples

```

trainReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Do not like it",
  "Really like it",
  "I hate it",
  "I like it a lot",
  "I kind of hate it",
  "I do like it",
  "I really hate it",
  "It is very good",
  "I hate it a bunch",
  "I love it a bunch",
  "I hate it",
  "I like it very much",
  "I hate it very much.",
  "I really do love it",
  "I really do hate it",
  "Love it!",
  "Hate it!",
  "I love it",
  "I hate it",
  "I love it",
  "I hate it",
  "I love it"),
  like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
)

```

```

testReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Really like it",
  "I hate it",
  "I like it a lot",
  "I love it",
  "I do like it",
  "I really hate it",
  "I love it"), stringsAsFactors = FALSE)

```

```

# Use a categorical transform: the entire string is treated as a category
outModel1 <- rxLogisticRegression(like~reviewCat, data = trainReviews,
  mlTransforms = list(categorical(vars = c(reviewCat = "review"))))
# Note that 'I hate it' and 'I love it' (the only strings appearing more
than once)
# have non-zero weights
summary(outModel1)

```

```
# Use the model to score
scoreOutDF1 <- rxPredict(outModel1, data = testReviews,
  extraVarsToWrite = "review")
scoreOutDF1
```

categoricalHash: Machine Learning Categorical HashData Transform

Article • 02/28/2023

Categorical hash transform that can be performed on data before training a model.

Usage

```
categoricalHash(vars, hashBits = 16, seed = 314489979, ordered = TRUE,
  invertHash = 0, outputKind = "Bag", ...)
```

Arguments

vars

A character vector or list of variable names to transform. If named, the names represent the names of new variables to be created.

hashBits

An integer specifying the number of bits to hash into. Must be between 1 and 30, inclusive. The default value is 16.

seed

An integer specifying the hashing seed. The default value is 314489979.

ordered

`TRUE` to include the position of each term in the hash. Otherwise, `FALSE`. The default value is `TRUE`.

invertHash

An integer specifying the limit on the number of keys that can be used to generate the slot name. `0` means no invert hashing; `-1` means no limit. While a zero value gives better performance, a non-zero value is needed to get meaningful coefficient names. The default value is `0`.

outputKind

A character string that specifies the kind of output kind.

- `"ind"`: Outputs an indicator vector. The input column is a vector of categories, and the output contains one indicator vector per slot in the input column.
- `"bag"`: Outputs a multi-set vector. If the input column is a vector of categories, the output contains one vector, where the value in each slot is the number of occurrences of the category in the input vector. If the input column contains a single category, the indicator vector and the bag vector are equivalent
- `"key"`: Outputs an index. The output is an integer ID (between 1 and the number of categories in the dictionary) of the category.
The default value is `"Bag"`.

...

Additional arguments sent to the compute engine.

Details

`categoricalHash` converts a categorical value into an indicator array by hashing the value and using the hash as an index in the bag. If the input column is a vector, a single indicator bag is returned for it.

`categoricalHash` does not currently support handling factor data.

Value

a `maml` object defining the transform.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

See also

[rxFastTrees](#), [rxFastForest](#), [rxNeuralNet](#), [rxOneClassSvm](#), [rxLogisticRegression](#).

Examples

```
trainReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Do not like it",
  "Really like it",
  "I hate it",
  "I like it a lot",
  "I kind of hate it",
  "I do like it",
  "I really hate it",
  "It is very good",
  "I hate it a bunch",
  "I love it a bunch",
  "I hate it",
  "I like it very much",
  "I hate it very much.",
  "I really do love it",
  "I really do hate it",
  "Love it!",
  "Hate it!",
  "I love it",
  "I hate it",
  "I love it",
  "I hate it",
  "I love it"),
  like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
)

testReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Really like it",
  "I hate it",
  "I like it a lot",
  "I love it",
  "I do like it",
  "I really hate it",
  "I love it"), stringsAsFactors = FALSE)
```

```
# Use a categorical hash transform
outModel2 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
  mlTransforms = list(categoricalHash(vars = c(reviewCatHash =
"review"))))
# Weights are similar to categorical
summary(outModel2)

# Use the model to score
scoreOutDF2 <- rxPredict(outModel2, data = testReviews,
  extraVarsToWrite = "review")
scoreOutDF2
```

concat: Machine Learning Concat Transform

Article • 02/28/2023

Combines several columns into a single vector-valued column.

Usage

```
concat(vars, ...)
```

Arguments

vars

A named list of character vectors of input variable names and the name of the output variable. Note that all the input variables must be of the same type. It is possible to produce multiple output columns with the concatenation transform. In this case, you need to use a list of vectors to define a one-to-one mapping between input and output variables. For example, to concatenate columns InNameA and InNameB into column OutName1 and also columns InNameC and InNameD into column OutName2, use the list: `(list(OutName1 = c(InNameA, InNameB), outName2 = c(InNameC, InNameD)))`

...

Additional arguments sent to the compute engine

Details

`concat` creates a single vector-valued column from multiple columns. It can be performed on data before training a model. The concatenation can significantly speed up the processing of data when the number of columns is as large as hundreds to thousands.

Value

A `maml` object defining the concatenation transform.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

See also

[featurizeText](#), [categorical](#), [categoricalHash](#), [rxFastTrees](#), [rxFastForest](#), [rxNeuralNet](#), [rxOneClassSvm](#), [rxLogisticRegression](#).

Examples

```
testObs <- rnorm(nrow(iris)) > 0
testIris <- iris[testObs,]
trainIris <- iris[!testObs,]

multiLogitOut <- rxLogisticRegression(
  formula = Species~Features, type = "multiClass", data = trainIris,
  mlTransforms = list(concat(vars = list(
    Features = c("Sepal.Length", "Sepal.Width", "Petal.Length",
"Petal.Width")
  )))
summary(multiLogitOut)
```

dropColumns: Drops columns from the dataset

Article • 02/28/2023

Specified columns to drop from the dataset.

Usage

```
dropColumns(vars, ...)
```

Arguments

vars

A character vector or list of the names of the variables to drop.

...

Additional arguments sent to compute engine.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

ensembleControl: ensembleControl

Article • 02/28/2023

Control the parameters used to create an ensemble.

Usage

```
ensembleControl(randomSeed = NULL, modelCount = 1, replace = FALSE,  
  sampRate = ifelse(replace, 1, 0.632), splitData = FALSE,  
  combineMethod = NULL, ...)
```

Arguments

randomSeed

Specifies the random seed. The default value is `NULL`.

modelCount

Specifies the number of models to train. The default value is `1`, meaning no ensembling occurs.

replace

A logical value specifying if the sampling of observations should be done with or without replacement. The default value is `FALSE`.

sampRate

a scalar of positive value specifying the percentage of observations to sample for each trainer. The default is 1.0 for sampling with replacement (i.e., `replace=TRUE`) and 0.632 for sampling without replacement (i.e., `replace=FALSE`).

splitData

A logical value that specifies whether or not to train the base models on non-overlapping partitions. The default is `FALSE`. It is available only for `RxSpark` compute context and is ignored for others.

`combineMethod`

Specifies the method used to combine the models:

- `median` to compute the median of the individual model outputs,
- `average` to compute the average of the individual model outputs and
- `vote` to compute $(\text{pos-neg}) / \text{the total number of models}$, where 'pos' is the number of positive outputs and 'neg' is the number of negative outputs. The default value is `median`.



Not used currently.

Value

A list of ensemble parameters.

extractPixels: Machine Learning Extract Pixel Data Transform

Article • 02/28/2023

Extracts the pixel values from an image.

Usage

```
extractPixels(vars, useAlpha = FALSE, useRed = TRUE, useGreen = TRUE,  
             useBlue = TRUE, interleaveARGB = FALSE, convert = TRUE, offset = NULL,  
             scale = NULL)
```

Arguments

vars

A named list of character vectors of input variable names and the name of the output variable. Note that the input variables must be of the same type. For one-to-one mappings between input and output variables, a named character vector can be used.

useAlpha

Specifies whether to use alpha channel. The default value is **FALSE**.

useRed

Specifies whether to use red channel. The default value is **TRUE**.

useGreen

Specifies whether to use green channel. The default value is **TRUE**.

useBlue

Specifies whether to use blue channel. The default value is `TRUE`.

`interleaveARGB`

Whether to separate each channel or interleave in ARGB order. This might be important, for example, if you are training a convolutional neural network, since this would affect the shape of the kernel, stride etc.

`convert`

Whether to convert to floating point. The default value is `FALSE`.

`offset`

Specifies the offset (pre-scale). This requires `convert = TRUE`. The default value is `NULL`.

`scale`

Specifies the scale factor. This requires `convert = TRUE`. The default value is `NULL`.

Details

`extractPixels` extracts the pixel values from an image. The input variables are images of the same size, typically the output of a `resizeImage` transform. The output is pixel data in vector form that are typically used as features for a learner.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) [↗](#)

Examples

```

train <- data.frame(Path =
c(system.file("help/figures/RevolutionAnalyticslogo.png", package =
"MicrosoftML")), Label = c(TRUE), stringsAsFactors = FALSE)

# Loads the images from variable Path, resizes the images to 1x1 pixels and
trains a neural net.
model <- rxNeuralNet(
  Label ~ Features,
  data = train,
  mlTransforms = list(
    loadImage(vars = list(Features = "Path")),
    resizeImage(vars = "Features", width = 1, height = 1, resizing =
"Aniso"),
    extractPixels(vars = "Features")
  ),
  mlTransformVars = "Path",
  numHiddenNodes = 1,
  numIterations = 1)

# Featurizes the images from variable Path using the default model, and
trains a linear model on the result.
model <- rxFastLinear(
  Label ~ Features,
  data = train,
  mlTransforms = list(
    loadImage(vars = list(Features = "Path")),
    resizeImage(vars = "Features", width = 224, height = 224), # If
dnnModel == "AlexNet", the image has to be resized to 227x227.
    extractPixels(vars = "Features"),
    featurizeImage(var = "Features")
  ),
  mlTransformVars = "Path")

```

fastForest: fastForest

Article • 02/28/2023

Creates a list containing the function name and arguments to train a FastForest model with [rxEnsemble](#).

Usage

```
fastForest(numTrees = 100, numLeaves = 20, minSplit = 10,  
           exampleFraction = 0.7, featureFraction = 0.7, splitFraction = 0.7,  
           numBins = 255, firstUsePenalty = 0, gainConfLevel = 0,  
           trainThreads = 8, randomSeed = NULL, ...)
```

Arguments

numTrees

Specifies the total number of decision trees to create in the ensemble. By creating more decision trees, you can potentially get better coverage, but the training time increases. The default value is 100.

numLeaves

The maximum number of leaves (terminal nodes) that can be created in any tree. Higher values potentially increase the size of the tree and get better precision, but risk overfitting and requiring longer training times. The default value is 20.

minSplit

Minimum number of training instances required to form a leaf. That is, the minimal number of documents allowed in a leaf of a regression tree, out of the sub-sampled data. A 'split' means that features in each level of the tree (node) are randomly divided. The default value is 10.

exampleFraction

The fraction of randomly chosen instances to use for each tree. The default value is 0.7.

featureFraction

The fraction of randomly chosen features to use for each tree. The default value is 0.7.

splitFraction

The fraction of randomly chosen features to use on each split. The default value is 0.7.

numBins

Maximum number of distinct values (bins) per feature. The default value is 255.

firstUsePenalty

The feature first use penalty coefficient. The default value is 0.

gainConfLevel

Tree fitting gain confidence requirement (should be in the range [0,1)). The default value is 0.

trainThreads

The number of threads to use in training. If `NULL` is specified, the number of threads to use is determined internally. The default value is `NULL`.

randomSeed

Specifies the random seed. The default value is `NULL`.



Additional arguments.

fastLinear: fastLinear

Article • 02/28/2023

Creates a list containing the function name and arguments to train a Fast Linear model with [rxEnsemble](#).

Usage

```
fastLinear(lossFunction = NULL, l2Weight = NULL, l1Weight = NULL,  
  trainThreads = NULL, convergenceTolerance = 0.1, maxIterations = NULL,  
  shuffle = TRUE, checkFrequency = NULL, ...)
```

Arguments

lossFunction

Specifies the empirical loss function to optimize. For binary classification, the following choices are available:

- [logLoss](#): The log-loss. This is the default.
- [hingeLoss](#): The SVM hinge loss. Its parameter represents the margin size.
- [smoothHingeLoss](#): The smoothed hinge loss. Its parameter represents the smoothing constant.

For linear regression, squared loss [squaredLoss](#) is currently supported. When this parameter is set to `NULL`, its default value depends on the type of learning:

- [logLoss](#) for binary classification.
- [squaredLoss](#) for linear regression.

l2Weight

Specifies the L2 regularization weight. The value must be either non-negative or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. `NULL` is the default value.

l1Weight

Specifies the L1 regularization weight. The value must be either non-negative or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. `NULL` is the default value.

trainThreads

Specifies how many concurrent threads can be used to run the algorithm. When this parameter is set to `NULL`, the number of threads used is determined based on the number of logical processors available to the process as well as the sparsity of data. Set it to `1` to run the algorithm in a single thread.

convergenceTolerance

Specifies the tolerance threshold used as a convergence criterion. It must be between 0 and 1. The default value is `0.1`. The algorithm is considered to have converged if the relative duality gap, which is the ratio between the duality gap and the primal loss, falls below the specified convergence tolerance.

maxIterations

Specifies an upper bound on the number of training iterations. This parameter must be positive or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. Each iteration requires a complete pass over the training data. Training terminates after the total number of iterations reaches the specified upper bound or when the loss function converges, whichever happens earlier.

shuffle

Specifies whether to shuffle the training data. Set `TRUE` to shuffle the data; `FALSE` not to shuffle. The default value is `TRUE`. SDCA is a stochastic optimization algorithm. If shuffling is turned on, the training data is shuffled on each iteration.

checkFrequency

The number of iterations after which the loss function is computed and checked to determine whether it has converged. The value specified must be a positive integer or `NULL`. If `NULL`, the actual value is automatically computed based on data set. Otherwise, for example, if `checkFrequency = 5` is specified, then the loss function is computed and

convergence is checked every 5 iterations. The computation of the loss function requires a separate complete pass over the training data.



Additional arguments.

fastTrees: fastTrees

Article • 02/28/2023

Creates a list containing the function name and arguments to train a FastTree model with [rxEnsemble](#).

Usage

```
fastTrees(numTrees = 100, numLeaves = 20, learningRate = 0.2,  
          minSplit = 10, exampleFraction = 0.7, featureFraction = 1,  
          splitFraction = 1, numBins = 255, firstUsePenalty = 0,  
          gainConfLevel = 0, unbalancedSets = FALSE, trainThreads = 8,  
          randomSeed = NULL, ...)
```

Arguments

numTrees

Specifies the total number of decision trees to create in the ensemble. By creating more decision trees, you can potentially get better coverage, but the training time increases. The default value is 100.

numLeaves

The maximum number of leaves (terminal nodes) that can be created in any tree. Higher values potentially increase the size of the tree and get better precision, but risk overfitting and requiring longer training times. The default value is 20.

learningRate

Determines the size of the step taken in the direction of the gradient in each step of the learning process. This determines how fast or slow the learner converges on the optimal solution. If the step size is too big, you might overshoot the optimal solution. If the step size is too small, training takes longer to converge to the best solution.

minSplit

Minimum number of training instances required to form a leaf. That is, the minimal number of documents allowed in a leaf of a regression tree, out of the sub-sampled data. A 'split' means that features in each level of the tree (node) are randomly divided. The default value is 10. Only the number of instances is counted even if instances are weighted.

exampleFraction

The fraction of randomly chosen instances to use for each tree. The default value is 0.7.

featureFraction

The fraction of randomly chosen features to use for each tree. The default value is 1.

splitFraction

The fraction of randomly chosen features to use on each split. The default value is 1.

numBins

Maximum number of distinct values (bins) per feature. If the feature has fewer values than the number indicated, each value is placed in its own bin. If there are more values, the algorithm creates `numBins` bins.

firstUsePenalty

The feature first use penalty coefficient. This is a form of regularization that incurs a penalty for using a new feature when creating the tree. Increase this value to create trees that don't use many features. The default value is 0.

gainConfLevel

Tree fitting gain confidence requirement (should be in the range [0,1)). The default value is 0.

unbalancedSets

If `TRUE`, derivatives optimized for unbalanced sets are used. Only applicable when `type` equal to `"binary"`. The default value is `FALSE`.

trainThreads

The number of threads to use in training. The default value is 8.

randomSeed

Specifies the random seed. The default value is `NULL`.



Additional arguments.

featurizeImage: Machine Learning Image Featurization Transform

Article • 02/28/2023

Featurizes an image using a pre-trained deep neural network model.

Usage

```
featurizeImage(var, outVar = NULL, dnnModel = "Resnet18")
```

Arguments

var

Input variable containing extracted pixel values.

outVar

The prefix of the output variables containing the image features. If null, the input variable name will be used. The default value is `NULL`.

dnnModel

The pre-trained deep neural network. The possible options are:

- `"resnet18"`
- `"resnet50"`
- `"resnet101"`
- `"alexnet"`

The default value is `"resnet18"`. See [Deep Residual Learning for Image Recognition](#) [↗](#) for details about ResNet.

Details

`featurizeImage` featurizes an image using the specified pre-trained deep neural network model. The input variables to this transform must be extracted pixel values.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

Examples

```
train <- data.frame(Path =
c(system.file("help/figures/RevolutionAnalyticslogo.png", package =
"MicrosoftML")), Label = c(TRUE), stringsAsFactors = FALSE)

# Loads the images from variable Path, resizes the images to 1x1 pixels and
trains a neural net.
model <- rxNeuralNet(
  Label ~ Features,
  data = train,
  mlTransforms = list(
    loadImage(vars = list(Features = "Path")),
    resizeImage(vars = "Features", width = 1, height = 1, resizing =
"Aniso"),
    extractPixels(vars = "Features")
  ),
  mlTransformVars = "Path",
  numHiddenNodes = 1,
  numIterations = 1)

# Featurizes the images from variable Path using the default model, and
trains a linear model on the result.
model <- rxFastLinear(
  Label ~ Features,
  data = train,
  mlTransforms = list(
    loadImage(vars = list(Features = "Path")),
    resizeImage(vars = "Features", width = 224, height = 224), # If
dnnModel == "AlexNet", the image has to be resized to 227x227.
    extractPixels(vars = "Features"),
    featurizeImage(var = "Features")
  ),
  mlTransformVars = "Path")
```


stopwordsDefault: Machine Learning Text Transform

Article • 02/28/2023

Text transforms that can be performed on data before training a model.

Usage

```
stopwordsDefault()

stopwordsCustom(dataFile = "")

termDictionary(terms = "", dataFile = "", sort = "occurrence")

featurizeText(vars, language = "English", stopwordsRemover = NULL,
  case = "lower", keepDiacritics = FALSE, keepPunctuations = TRUE,
  keepNumbers = TRUE, dictionary = NULL,
  wordFeatureExtractor = ngramCount(), charFeatureExtractor = NULL,
  vectorNormalizer = "l2", ...)
```

Arguments

dataFile

character: <string>. Data file containing the terms (short form data).

terms

An optional character vector of terms or categories.

sort

Specifies how to order items when vectorized. Two orderings are supported:

- **"occurrence"**: items appear in the order encountered.
- **"value"**: items are sorted according to their default comparison. For example, text sorting will be case sensitive (e.g., 'A' then 'Z' then 'a').

vars

A named list of character vectors of input variable names and the name of the output variable. Note that the input variables must be of the same type. For one-to-one mappings between input and output variables, a named character vector can be used.

language

Specifies the language used in the data set. The following values are supported:

- "AutoDetect": for automatic language detection.
- "English".
- "French".
- "German".
- "Dutch".
- "Italian".
- "Spanish".
- "Japanese".

stopwordsRemover

Specifies the stopwords remover to use. There are three options supported:

- `NULL` No stopwords remover is used.
- `stopwordsDefault`: A precompiled language-specific list of stop words is used that includes the most common words from Microsoft Office.
- `stopwordsCustom`: A user-defined list of stopwords. It accepts the following option:
`dataFile`.

The default value is `NULL`.

case

Text casing using the rules of the invariant culture. Takes the following values:

- "lower".
- "upper".
- "none".

The default value is "lower".

keepDiacritics

`FALSE` to remove diacritical marks; `TRUE` to retain diacritical marks. The default value is `FALSE`.

keepPunctuations

`FALSE` to remove punctuation; `TRUE` to retain punctuation. The default value is `TRUE`.

keepNumbers

`FALSE` to remove numbers; `TRUE` to retain numbers. The default value is `TRUE`.

dictionary

A `termDictionary` of allowlisted terms which accepts the following options:

- `terms`,
- `dataFile`, and
- `sort`.

The default value is `NULL`. Note that the stopwords list takes precedence over the dictionary allowlist as the stopwords are removed before the dictionary terms are allowlisted.

wordFeatureExtractor

Specifies the word feature extraction arguments. There are two different feature extraction mechanisms:

- `ngramCount`: Count-based feature extraction (equivalent to `WordBag`). It accepts the following options: `maxNumTerms` and `weighting`.
- `ngramHash`: Hashing-based feature extraction (equivalent to `WordHashBag`). It accepts the following options: `hashBits`, `seed`, `ordered` and `invertHash`.

The default value is `ngramCount`.

charFeatureExtractor

Specifies the char feature extraction arguments. There are two different feature extraction mechanisms:

- `ngramCount`: Count-based feature extraction (equivalent to `WordBag`). It accepts the following options: `maxNumTerms` and `weighting`.

- `ngramHash`: Hashing-based feature extraction (equivalent to `WordHashBag`). It accepts the following options: `hashBits`, `seed`, `ordered` and `invertHash`. The default value is `NULL`.

`vectorNormalizer`

Normalize vectors (rows) individually by rescaling them to unit norm. Takes one of the following values:

- `"none"`.
- `"l2"`.
- `"l1"`.
- `"l1nf"`. The default value is `"l2"`.



Additional arguments sent to the compute engine.

Details

The `featurizeText` transform produces a bag of counts of sequences of consecutive words, called n-grams, from a given corpus of text. There are two ways it can do this:

build a dictionary of n-grams and use the ID in the dictionary as the index in the bag;

hash each n-gram and use the hash value as the index in the bag.

The purpose of hashing is to convert variable-length text documents into equal-length numeric feature vectors, to support dimensionality reduction and to make the lookup of feature weights faster.

The text transform is applied to text input columns. It offers language detection, tokenization, stopwords removing, text normalization and feature generation. It supports the following languages by default: English, French, German, Dutch, Italian, Spanish and Japanese.

The n-grams are represented as count vectors, with vector slots corresponding either to n-grams (created using `ngramCount`) or to their hashes (created using `ngramHash`). Embedding ngrams in a vector space allows their contents to be compared in an efficient manner. The slot values in the vector can be weighted by the following factors:

term frequency - The number of occurrences of the slot in the text

inverse document frequency - A ratio (the logarithm of inverse relative slot frequency) that measures the information a slot provides by determining how common or rare it is across the entire text.

term frequency-inverse document frequency - the product term frequency and the inverse document frequency.

Value

A `map1` object defining the transform.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

See also

[ngramCount](#), [ngramHash](#), [rxFastTrees](#), [rxFastForest](#), [rxNeuralNet](#), [rxOneClassSvm](#), [rxLogisticRegression](#).

Examples

```
trainReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Do not like it",
  "Really like it",
  "I hate it",
  "I like it a lot",
  "I kind of hate it",
  "I do like it",
  "I really hate it",
  "It is very good",
  "I hate it a bunch",
  "I love it a bunch",
  "I hate it",
  "I like it very much",
  "I hate it very much.",
  "I really do love it",
```

```

    "I really do hate it",
    "Love it!",
    "Hate it!",
    "I love it",
    "I hate it",
    "I love it",
    "I hate it",
    "I love it"),
  like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
           FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
           FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
           FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
)

```

```

testReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Really like it",
  "I hate it",
  "I like it a lot",
  "I love it",
  "I do like it",
  "I really hate it",
  "I love it"), stringsAsFactors = FALSE)

```

```

outModel <- rxLogisticRegression(like ~ reviewTran, data = trainReviews,
  mlTransforms = list(featurizeText(vars = c(reviewTran = "review"),
  stopwordsRemover = stopwordsDefault(), keepPunctuations = FALSE)))
# 'hate' and 'love' have non-zero weights
summary(outModel)

# Use the model to score
scoreOutDF5 <- rxPredict(outModel, data = testReviews,
  extraVarsToWrite = "review")
scoreOutDF5

```


getNetDefinition: Get the Net# definition from a trained neural network model

Article • 02/28/2023

Returns the Net# definition from a trained neural network model.

Usage

```
getNetDefinition(model, getWeights = TRUE)
```

Arguments

model

The previously trained neural network model.

getWeights

If `TRUE`, the weights are included in the returned Net# definition.

Details

Returns the Net# definition from a trained neural network model. It is useful for implementing a form of continued training, where the initial weights of the model are obtained from a previously trained model. Because only the weights are initialized from the trained model (but not gradients, momentum etc.), the training is not resumed where it was left at the end of training of the first model.

Value

A character string containing the Net# definition.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

Examples

```
# Train a neural network on the iris dataset for 10 iterations.
model1 <- rxNeuralNet(
  formula = Species~Sepal.Length + Sepal.Width + Petal.Length +
Petal.Width,
  data = iris,
  numHiddenNodes=10,
  type="multi",
  numIterations=10,
  optimizer=adaDeltaSgd())

# Train another neural network on the iris dataset, initializing the
topology and weights
# from the previously trained model.
model2 <- rxNeuralNet(
  formula = Species~Sepal.Length + Sepal.Width + Petal.Length +
Petal.Width,
  data = iris,
  netDefinition=getNetDefinition(model1),
  type="multi",
  numIterations=10,
  optimizer = adaDeltaSgd())
```

getSampleDataDir: Get Package Sample Data Location

Article • 02/28/2023

Location where downloaded sample data is stored.

Usage

```
getSampleDataDir(sampleDataDir = NULL, createDir = TRUE)
```

Arguments

`sampleDataDir`

Specifies the path to the location where downloaded sample data is (or is to be) stored or `NULL`.

`createDir`

`TRUE` to create the directory if it does not exist; `FALSE` not to create the directory.

Details

If `sampleDataDir` is `NULL`, the function first checks to see if an option has been set containing `sampleDataDir`, i.e. `getOption("sampleDataDir")`. If that is `NULL` too, a 'sampleDataDir' subdirectory of the current working directory is used. If `createDir` is `TRUE`, the directory is created if it does not exist.

Value

A character string containing the path to the location of the sample data.

Author(s)

Examples

```
# This example sets the option to be the same as the default
options(sampleDataDir = file.path(getwd(), "sampleDataDir"))

dataDir <- getSampleDataDir(createDir = FALSE)
```

getSentiment: Machine Learning Sentiment Analyzer Transform

Article • 02/28/2023

Scores natural language text and creates a column that contains probabilities that the sentiments in the text are positive.

Usage

```
getSentiment(vars, ...)
```

Arguments

vars

A character vector or list of variable names to transform. If named, the names represent the names of new variables to be created.

...

Additional arguments sent to compute engine.

Details

The `getSentiment` transform returns the probability that the sentiment of a natural text is positive. Currently supports only the English language.

Value

A `maml` object defining the transform.

Author(s)

See also

[rxFastTrees](#), [rxFastForest](#), [rxNeuralNet](#), [rxOneClassSvm](#), [rxLogisticRegression](#), [rxFastLinear](#).

Examples

```
# Create the data
CustomerReviews <- data.frame(Review = c(
  "I really did not like the taste of it",
  "It was surprisingly quite good!",
  "I will never ever ever go to that place again!!"),
  stringsAsFactors = FALSE)

# Get the sentiment scores
sentimentScores <- rxFeaturize(data = CustomerReviews,
                              mlTransforms = getSentiment(vars =
list(SentimentScore = "Review")))

# Let's translate the score to something more meaningful
sentimentScores$PredictedRating <- ifelse(sentimentScores$SentimentScore >
0.6,
                                         "AWESOMENESS", "BLAH")

# Let's look at the results
sentimentScores
```

kernel: Kernel

Article • 02/28/2023

Kernels supported for use in computing inner products.

Usage

```
linearKernel(...)  
  
polynomialKernel(a = NULL, bias = 0, deg = 3, ...)  
  
rbfKernel(gamma = NULL, ...)  
  
sigmoidKernel(gamma = NULL, coef0 = 0, ...)
```

Arguments

a

The numeric value for a in the term $(a \cdot \langle x, y \rangle + b)^d$. If not specified, $1/(\text{number of features})$ is used.

bias

The numeric value for b in the term $(a \cdot \langle x, y \rangle + b)^d$.

deg

The integer value for d in the term $(a \cdot \langle x, y \rangle + b)^d$.

gamma

The numeric value for gamma in the expression $\tanh(\text{gamma} \cdot \langle x, y \rangle + c)$. If not specified, $1/(\text{number of features})$ is used.

coef0

The numeric value for c in the expression `tanh(gamma*<x,y> + c)`.



Additional arguments passed to the Microsoft ML compute engine.

Details

These helper functions specify the kernel that is used for training in relevant algorithms. The kernels that are supported:

`linearKernel`: linear kernel.

`rbfKernel`: radial basis function kernel.

`polynomialKernel`: polynomial kernel.

`sigmoidKernel`: sigmoid kernel.

Value

A character string defining the kernel.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

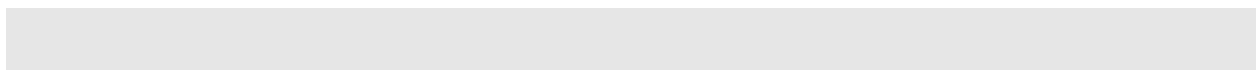
[Estimating the Support of a High-Dimensional Distribution](#)

[New Support Vector Algorithms](#)

See also

[rxOneClassSvm](#)

Examples




```

# Simulate some simple data
set.seed(7)
numRows <- 200
normalData <- data.frame(day = 1:numRows)
normalData$pageViews = runif(numRows, min = 10, max = 1000) + .5 *
normalData$day
testData <- data.frame(day = 1:numRows)
# The test data has outliers above 1000
testData$pageViews = runif(numRows, min = 10, max = 1400) + .5 *
testData$day

train <- function(kernelFunction, args=NULL) {
  model <- rxOneClassSvm(formula = ~pageViews + day, data = normalData,
    kernel = kernelFunction(args))
  scores <- rxPredict(model, data = testData, writeModelVars = TRUE)
  scores$groups = scores$Score > 0
  scores
}

display <- function(scores) {
  print(sum(scores$groups))
  rxLinePlot(pageViews ~ day, data = scores, groups = groups, type = "p",
    symbolColors = c("red", "blue"))
}

scores <- list()
scores$rbfKernel <- train(rbfKernel)
scores$linearKernel <- train(linearKernel)
scores$polynomialKernel <- train(polynomialKernel, (a = .2))
scores$sigmoidKernel <- train(sigmoidKernel)
display(scores$rbfKernel)
display(scores$linearKernel)
display(scores$polynomialKernel)
display(scores$sigmoidKernel)

```

loadImage: Machine Learning Load Image Transform

Article • 02/28/2023

Loads image data.

Usage

```
loadImage(vars)
```

Arguments

vars

A named list of character vectors of input variable names and the name of the output variable. Note that the input variables must be of the same type. For one-to-one mappings between input and output variables, a named character vector can be used.

Details

`loadImage` loads images from paths.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) ↗

Examples

```

train <- data.frame(Path =
c(system.file("help/figures/RevolutionAnalyticslogo.png", package =
"MicrosoftML")), Label = c(TRUE), stringsAsFactors = FALSE)

# Loads the images from variable Path, resizes the images to 1x1 pixels and
trains a neural net.
model <- rxNeuralNet(
  Label ~ Features,
  data = train,
  mlTransforms = list(
    loadImage(vars = list(Features = "Path")),
    resizeImage(vars = "Features", width = 1, height = 1, resizing =
"Aniso"),
    extractPixels(vars = "Features")
  ),
  mlTransformVars = "Path",
  numHiddenNodes = 1,
  numIterations = 1)

# Featurizes the images from variable Path using the default model, and
trains a linear model on the result.
model <- rxFastLinear(
  Label ~ Features,
  data = train,
  mlTransforms = list(
    loadImage(vars = list(Features = "Path")),
    resizeImage(vars = "Features", width = 224, height = 224), # If
dnnModel == "AlexNet", the image has to be resized to 227x227.
    extractPixels(vars = "Features"),
    featurizeImage(var = "Features")
  ),
  mlTransformVars = "Path")

```

logisticRegression: logisticRegression

Article • 02/28/2023

Creates a list containing the function name and arguments to train a logistic regression model with [rxEnsemble](#).

Usage

```
logisticRegression(l2Weight = 1, l1Weight = 1, optTol = 1e-07,  
memorySize = 20, initWtsScale = 0, maxIterations = 2147483647,  
showTrainingStats = FALSE, sgdInitTol = 0, trainThreads = NULL,  
denseOptimizer = FALSE, ...)
```

Arguments

l2Weight

The L2 regularization weight. Its value must be greater than or equal to `0` and the default value is set to `1`.

l1Weight

The L1 regularization weight. Its value must be greater than or equal to `0` and the default value is set to `1`.

optTol

Threshold value for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model. Smaller values are slower, but more accurate. The default value is `1e-07`.

memorySize

Memory size for L-BFGS, specifying the number of past positions and gradients to store for the computation of the next step. This optimization parameter limits the amount of

memory that is used to compute the magnitude and direction of the next step. When you specify less memory, training is faster but less accurate. Must be greater than or equal to `1` and the default value is `20`.

initWtsScale

Sets the initial weights diameter that specifies the range from which values are drawn for the initial weights. These weights are initialized randomly from within this range. For example, if the diameter is specified to be `d`, then the weights are uniformly distributed between `-d/2` and `d/2`. The default value is `0`, which specifies that all the weights are initialized to `0`.

maxIterations

Sets the maximum number of iterations. After this number of steps, the algorithm stops even if it has not satisfied convergence criteria.

showTrainingStats

Specify `TRUE` to show the statistics of training data and the trained model; otherwise, `FALSE`. The default value is `FALSE`. For additional information about model statistics, see [summary.mlModel](#).

sgdInitTol

Set to a number greater than 0 to use Stochastic Gradient Descent (SGD) to find the initial parameters. A non-zero value set specifies the tolerance SGD uses to determine convergence. The default value is `0` specifying that SGD is not used.

trainThreads

The number of threads to use in training the model. This should be set to the number of cores on the machine. Note that L-BFGS multi-threading attempts to load dataset into memory. In case of out-of-memory issues, set `trainThreads` to `1` to turn off multi-threading. If `NULL` the number of threads to use is determined internally. The default value is `NULL`.

denseOptimizer

If `TRUE`, forces densification of the internal optimization vectors. If `FALSE`, enables the logistic regression optimizer use sparse or dense internal states as it finds appropriate. Setting `denseOptimizer` to `TRUE` requires the internal optimizer to use a dense internal state, which may help alleviate load on the garbage collector for some varieties of larger problems.



Additional arguments.

loss functions: Classification and Regression Loss functions

Article • 02/28/2023

The loss functions for classification and regression.

Usage

```
expLoss(beta = 1, ...)  
hingeLoss(margin = 1, ...)  
logLoss(...)  
smoothHingeLoss(smoothingConst = 1, ...)  
poissonLoss(...)  
squaredLoss(...)
```

Arguments

beta

Specifies the numeric value of beta (dilation). The default value is 1.

margin

Specifies the numeric margin value. The default value is 1.

smoothingConst

Specifies the numeric value of the smoothing constant. The default value is 1.

...

hidden argument.

Details

A loss function measures the discrepancy between the prediction of a machine learning algorithm and the supervised output and represents the cost of being wrong.

The classification loss functions supported are:

`logLoss`

`expLoss`

`hingeLoss`

`smoothHingeLoss`

The regression loss functions supported are:

`poissonLoss`

`squaredLoss`.

Value

A character string defining the loss function.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

See also

[rxFastLinear](#), [rxNeuralNet](#)

Examples

```
train <- function(lossFunction) {  
  result <- rxFastLinear(isCase ~ age + parity + education + spontaneous  
+ induced,  
                        transforms = list(isCase = case == 1), lossFunction =
```



```
lossFunction,
      data = infert,
      type = "binary")
  coef(result)[["age"]]
}

age <- list()
age$LogLoss <- train(logLoss())
age$LogLossHinge <- train(hingeLoss())
age$LogLossSmoothHinge <- train(smoothHingeLoss())
age
```

minCount: Feature Selection Count Mode

Article • 02/28/2023

Count mode of feature selection used in the feature selection transform [selectFeatures](#).

Usage

```
minCount(count = 1, ...)
```

Arguments

count

The threshold for count-based feature selection. A feature is selected if and only if at least `count` examples have non-default value in the feature. The default value is 1.

...

Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

When using the count mode in feature selection transform, a feature is selected if the number of examples have at least the specified count examples of non-default values in the feature. The count mode feature selection transform is useful when applied together with a categorical hash transform (see also, [categoricalHash](#)). The count feature selection can remove those features generated by hash transform that have no data in the examples.

Value

A character string defining the count mode.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

See also

[mutualInformation selectFeatures](#)

Examples

```
trainReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Do not like it",
  "Really like it",
  "I hate it",
  "I like it a lot",
  "I kind of hate it",
  "I do like it",
  "I really hate it",
  "It is very good",
  "I hate it a bunch",
  "I love it a bunch",
  "I hate it",
  "I like it very much",
  "I hate it very much.",
  "I really do love it",
  "I really do hate it",
  "Love it!",
  "Hate it!",
  "I love it",
  "I hate it",
  "I love it",
  "I hate it",
  "I love it",
  "I hate it",
  "I love it"),
  like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
)

testReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Really like it",
```

```

    "I hate it",
    "I like it a lot",
    "I love it",
    "I do like it",
    "I really hate it",
    "I love it"), stringsAsFactors = FALSE)

# Use a categorical hash transform which generated 128 features.
outModel1 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
l1Weight = 0,
  mlTransforms = list(categoricalHash(vars = c(reviewCatHash = "review"),
hashBits = 7)))
summary(outModel1)

# Apply a categorical hash transform and a count feature selection
transform
# which selects only those hash features that has value.
outModel2 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
l1Weight = 0,
  mlTransforms = list(
    categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
    selectFeatures("reviewCatHash", mode = minCount()))))
summary(outModel2)

# Apply a categorical hash transform and a mutual information feature
selection transform
# which selects those features appearing with at least a count of 5.
outModel3 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
l1Weight = 0,
  mlTransforms = list(
    categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
    selectFeatures("reviewCatHash", mode = minCount(count = 5))))
summary(outModel3)

```

summary.mlModel: Summary of a Microsoft R Machine Learning model.

Article • 02/28/2023

Summary of a Microsoft R Machine Learning model.

Usage

```
## S3 method for class `mlModel':  
summary (object, top = 20, ...)
```

Arguments

object

A model object returned from a **MicrosoftML** analysis.

top

Specifies the count of top coefficients to show in the summary for linear models such as [rxLogisticRegression](#) and [rxFastLinear](#). The bias appears first, followed by other weights, sorted by their absolute values in descending order. If set to `NULL`, all non-zero coefficients are shown. Otherwise, only the first `top` coefficients are shown.

...

Additional arguments to be passed to the summary method.

Details

Provides summary information about the original function call, the data set used to train the model, and statistics for coefficients in the model.

Value

The `summary` method of the **MicrosoftML** analysis objects returns a list that includes the original function call and the underlying parameters used. The `coef` method returns a named vector of weights, processing information from the model object.

For `rxLogisticRegression`, the following statistics may also present in the summary when `showTrainingStats` is set to `TRUE`.

`training.size`

The size, in terms of row count, of the data set used to train the model.

`deviance`

The model deviance is given by $-2 * \ln(L)$ where L is the likelihood of obtaining the observations with all features incorporated in the model.

`null.deviance`

The null deviance is given by $-2 * \ln(L_0)$ where L_0 is the likelihood of obtaining the observations with no effect from the features. The null model includes the bias if there is one in the model.

`aic`

The AIC (Akaike Information Criterion) is defined as $2 * k + deviance$, where k is the number of coefficients of the model. The bias counts as one of the coefficients. The AIC is a measure of the relative quality of the model. It deals with the trade-off between the goodness of fit of the model (measured by deviance) and the complexity of the model (measured by number of coefficients).

`coefficients.stats`

This is a data frame containing the statistics for each coefficient in the model. For each coefficient, the following statistics are shown. The bias appears in the first row, and the remaining coefficients in the ascending order of p-value.

- EstimateThe estimated coefficient value of the model.

- Std Error This is the square root of the large-sample variance of the estimate of the coefficient.
- z-Score We can test against the null hypothesis, which states that the coefficient should be zero, concerning the significance of the coefficient by calculating the ratio of its estimate and its standard error. Under the null hypothesis, if there is no regularization applied, the estimate of the concerning coefficient follows a normal distribution with mean 0 and a standard deviation equal to the standard error computed above. The z-score outputs the ratio between the estimate of a coefficient and the standard error of the coefficient.
- $P(>|z|)$ This is the corresponding p-value for the two-sided test of the z-score. Based on the significance level, a significance indicator is appended to the p-value. If $F(x)$ is the CDF of the standard normal distribution $N(0, 1)$, then $P(>|z|) = 2 * F(|z|)$.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

See also

[rxFastTrees](#), [rxFastForest](#), [rxFastLinear](#), [rxOneClassSvm](#), [rxNeuralNet](#), [rxLogisticRegression](#).

Examples

```
# Estimate a logistic regression model
logitModel <- rxLogisticRegression(isCase ~ age + parity + education +
spontaneous + induced,
                                transforms = list(isCase = case == 1),
                                data = infert)
# Print a summary of the model
summary(logitModel)

# Score to a data frame
scoreDF <- rxPredict(logitModel, data = infert,
                     extraVarsToWrite = "isCase")

# Compute and plot the Radio Operator Curve and AUC
roc1 <- rxRoc(actualVarName = "isCase", predVarNames = "Probability", data
= scoreDF)
plot(roc1)
rxAuc(roc1)
```

```
#####  
#####  
# Multi-class logistic regression  
testObs <- rnorm(nrow(iris)) > 0  
testIris <- iris[testObs,]  
trainIris <- iris[!testObs,]  
multiLogit <- rxLogisticRegression(  
  formula = Species~Sepal.Length + Sepal.Width + Petal.Length +  
Petal.Width,  
  type = "multiClass", data = trainIris)  
  
# Score the model  
scoreMultiDF <- rxPredict(multiLogit, data = testIris,  
  extraVarsToWrite = "Species")  
# Print the first rows of the data frame with scores  
head(scoreMultiDF)  
# Look at confusion matrix  
table(scoreMultiDF$Species, scoreMultiDF$PredictedLabel)  
  
# Look at the observations with incorrect predictions  
badPrediction = scoreMultiDF$Species != scoreMultiDF$PredictedLabel  
scoreMultiDF[badPrediction,]
```


mutualInformation: Feature Selection

Mutual Information Mode

Article • 02/28/2023

Mutual information mode of feature selection used in the feature selection transform [selectFeatures](#).

Usage

```
mutualInformation(numFeaturesToKeep = 1000, numBins = 256, ...)
```

Arguments

numFeaturesToKeep

If the number of features to keep is specified to be `n`, the transform picks the `n` features that have the highest mutual information with the dependent variable. The default value is 1000.

numBins

Maximum number of bins for numerical values. Powers of 2 are recommended. The default value is 256.

...

Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

The mutual information of two random variables `X` and `Y` is a measure of the mutual dependence between the variables. Formally, the mutual information can be written as:

$$I(X;Y) = E[\log(p(x,y)) - \log(p(x)) - \log(p(y))]$$

where the expectation is taken over the joint distribution of X and Y . Here $p(x,y)$ is the joint probability density function of X and Y , $p(x)$ and $p(y)$ are the marginal probability density functions of X and Y respectively. In general, a higher mutual information between the dependent variable (or label) and an independent variable (or feature) means that the label has higher mutual dependence over that feature.

The mutual information feature selection mode selects the features based on the mutual information. It keeps the top `numFeaturesToKeep` features with the largest mutual information with the label.

Value

a character string defining the mode.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) [↗](#)

References

[Wikipedia: Mutual Information](#) [↗](#)

See also

[minCount](#) [selectFeatures](#)

Examples

```
trainReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Do not like it",
  "Really like it",
  "I hate it",
  "I like it a lot",
  "I kind of hate it",
  "I do like it",
  "I really hate it",
```

```

    "It is very good",
    "I hate it a bunch",
    "I love it a bunch",
    "I hate it",
    "I like it very much",
    "I hate it very much.",
    "I really do love it",
    "I really do hate it",
    "Love it!",
    "Hate it!",
    "I love it",
    "I hate it",
    "I love it",
    "I hate it",
    "I love it"),
  like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
           FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
           FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
           FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
)

testReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Really like it",
  "I hate it",
  "I like it a lot",
  "I love it",
  "I do like it",
  "I really hate it",
  "I love it"), stringsAsFactors = FALSE)

# Use a categorical hash transform which generated 128 features.
outModel1 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
  l1Weight = 0,
  mlTransforms = list(categoricalHash(vars = c(reviewCatHash = "review"),
  hashBits = 7)))
summary(outModel1)

# Apply a categorical hash transform and a count feature selection
transform
# which selects only those hash features that has value.
outModel2 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
  l1Weight = 0,
  mlTransforms = list(
  categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
  selectFeatures("reviewCatHash", mode = minCount()))))
summary(outModel2)

# Apply a categorical hash transform and a mutual information feature
selection transform
# which selects those features appearing with at least a count of 5.
outModel3 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
  l1Weight = 0,

```

```
mlTransforms = list(  
  categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),  
  selectFeatures("reviewCatHash", mode = minCount(count = 5)))  
summary(outModel3)
```

neuralNet: neuralNet

Article • 03/01/2023

Creates a list containing the function name and arguments to train a NeuralNet model with [rxEnsemble](#).

Usage

```
neuralNet(numHiddenNodes = 100, numIterations = 100, optimizer = sgd(),
  netDefinition = NULL, initWtsDiameter = 0.1, maxNorm = 0,
  acceleration = c("sse", "gpu"), miniBatchSize = 1, ...)
```

Arguments

numHiddenNodes

The default number of hidden nodes in the neural net. The default value is 100.

numIterations

The number of iterations on the full training set. The default value is 100.

optimizer

A list specifying either the `sgd` or `adaptive` optimization algorithm. This list can be created using `sgd` or `adaDeltaSgd`. The default value is `sgd`.

netDefinition

The Net# definition of the structure of the neural network. For more information about the Net# language, see [Reference Guide](#)

initWtsDiameter

Sets the initial weights diameter that specifies the range from which values are drawn for the initial learning weights. The weights are initialized randomly from within this range. The default value is 0.1.

maxNorm

Specifies an upper bound to constrain the norm of the incoming weight vector at each hidden unit. This can be important in maxout neural networks and in cases where training produces unbounded weights.

acceleration

Specifies the type of hardware acceleration to use. Possible values are "sse" and "gpu". For GPU acceleration, it is recommended to use a miniBatchSize greater than one. If you want to use the GPU acceleration, there are additional manual setup steps are required:

- Download and install NVidia CUDA Toolkit 6.5 ([CUDA Toolkit](#)).
- Download and install NVidia cuDNN v2 Library ([cudnn Library](#)).
- Find the libs directory of the MicrosoftRML package by calling `system.file("mxLibs/x64", package = "MicrosoftML")`.
- Copy cublas64_65.dll, cudart64_65.dll and cusparse64_65.dll from the CUDA Toolkit 6.5 into the libs directory of the MicrosoftML package.
- Copy cudnn64_65.dll from the cuDNN v2 Library into the libs directory of the MicrosoftML package.

miniBatchSize

Sets the mini-batch size. Recommended values are between 1 and 256. This parameter is only used when the acceleration is GPU. Setting this parameter to a higher value improves the speed of training, but it might negatively affect the accuracy. The default value is 1.



Additional arguments.

ngram: Machine Learning Feature Extractors

Article • 02/28/2023

Feature Extractors that can be used with mtText.

Usage

```
ngramCount(ngramLength = 1, skipLength = 0, maxNumTerms = 1e+07,  
           weighting = "tf")  
  
ngramHash(ngramLength = 1, skipLength = 0, hashBits = 16,  
          seed = 314489979, ordered = TRUE, invertHash = 0)
```

Arguments

ngramLength

An integer that specifies the maximum number of tokens to take when constructing an n-gram. The default value is 1.

skipLength

An integer that specifies the maximum number of tokens to skip when constructing an n-gram. If the value specified as skip length is `k`, then n-grams can contain up to `k` skips (not necessarily consecutive). For example, if `k=2`, then the 3-grams extracted from the text "the sky is blue today" are: "the sky is", "the sky blue", "the sky today", "the is blue", "the is today" and "the blue today". The default value is 0.

maxNumTerms

An integer that specifies the maximum number of categories to include in the dictionary. The default value is 10000000.

weighting

A character string that specifies the weighting criteria:

- `"tf"`: to use term frequency.
- `"idf"`: to use inverse document frequency.
- `"tfidf"`: to use both term frequency and inverse document frequency.

`hashBits`

integer value. Number of bits to hash into. Must be between 1 and 30, inclusive.

`seed`

integer value. Hashing seed.

`ordered`

`TRUE` to include the position of each term in the hash. Otherwise, `FALSE`. The default value is `TRUE`.

`invertHash`

An integer specifying the limit on the number of keys that can be used to generate the slot name. `0` means no invert hashing; `-1` means no limit. While a zero value gives better performance, a non-zero value is needed to get meaningful coefficient names.

Details

`ngramCount` allows defining arguments for count-based feature extraction. It accepts following options: `ngramLength`, `skipLength`, `maxNumTerms` and `weighting`.

`ngramHash` allows defining arguments for hashing-based feature extraction. It accepts the following options: `ngramLength`, `skipLength`, `hashBits`, `seed`, `ordered` and `invertHash`.

Value

A character string defining the transform.

Author(s)

See also

[featurizeText](#).

Examples

```
myData <- data.frame(opinion = c(
  "I love it!",
  "I love it!",
  "Love it!",
  "I love it a lot!",
  "Really love it!",
  "I hate it",
  "I hate it",
  "I hate it.",
  "Hate it",
  "Hate"),
  like = rep(c(TRUE, FALSE), each = 5),
  stringsAsFactors = FALSE)

outModel1 <- rxLogisticRegression(like~opinionCount, data = myData,
  mlTransforms = list(featurizeText(vars = c(opinionCount = "opinion"),
    wordFeatureExtractor = ngramHash(invertHash = -1, hashBits = 3)))
summary(outModel1)

outModel2 <- rxLogisticRegression(like~opinionCount, data = myData,
  mlTransforms = list(featurizeText(vars = c(opinionCount = "opinion"),
    wordFeatureExtractor = ngramCount(maxNumTerms = 5, weighting =
"tf"))))
summary(outModel2)
```

oneClassSvm: oneClassSvm

Article • 02/28/2023

Creates a list containing the function name and arguments to train a OneClassSvm model with [rxEnsemble](#).

Usage

```
oneClassSvm(cacheSize = 100, kernel = rbfKernel(), epsilon = 0.001,  
            nu = 0.1, shrink = TRUE, ...)
```

Arguments

cacheSize

The maximal size in MB of the cache that stores the training data. Increase this for large training sets. The default value is 100 MB.

kernel

A character string representing the kernel used for computing inner products. For more information, see [maKernel](#). The following choices are available:

- `rbfKernel()`: Radial basis function kernel. Its parameter represents `gamma` in the term $\exp(-\text{gamma}|x-y|^2)$. If not specified, it defaults to `1` divided by the number of features used. For example, `rbfKernel(gamma = .1)`. This is the default value.
- `linearKernel()`: Linear kernel.
- `polynomialKernel()`: Polynomial kernel with parameter names `a`, `bias`, and `deg` in the term $(a \cdot \langle x, y \rangle + \text{bias})^{\text{deg}}$. The `bias`, defaults to `0`. The degree, `deg`, defaults to `3`. If `a` is not specified, it is set to `1` divided by the number of features. For example, `maKernelPoynomial(bias = 0, deg = `` 3)`.
- `sigmoidKernel()`: Sigmoid kernel with parameter names `gamma` and `coef0` in the term $\tanh(\text{gamma} \cdot \langle x, y \rangle + \text{coef0})$. `gamma`, defaults to `1` divided by the number of features. The parameter `coef0` defaults to `0`. For example, `sigmoidKernel(gamma = .1, coef0 = 0)`.

epsilon

The threshold for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model. The value must be greater than or equal to `.Machine$double.eps`. The default value is 0.001.

nu

The trade-off between the fraction of outliers and the number of support vectors (represented by the Greek letter nu). Must be between 0 and 1, typically between 0.1 and 0.5. The default value is 0.1.

shrink

Uses the shrinking heuristic if `TRUE`. In this case, some samples will be "shrunk" during the training procedure, which may speed up training. The default value is `TRUE`.

...

Additional arguments to be passed directly to the Microsoft Compute Engine.

maOptimizer: Optimization Algorithms

Article • 02/28/2023

Specifies Optimization Algorithms for Neural Net.

Usage

```
adaDeltaSgd(decay = 0.95, conditioningConst = 1e-06)

sgd(learningRate = 0.001, momentum = 0, nag = FALSE, weightDecay = 0,
    lRateRedRatio = 1, lRateRedFreq = 100, lRateRedErrorRatio = 0)
```

Arguments

decay

Specifies the decay rate applied to gradients when calculating the step in the ADADELTA adaptive optimization algorithm. This rate is used to ensure that the learning rate continues to make progress by giving smaller weights to remote gradients in the calculation of the step size. Mathematically, it replaces the mean square of the gradients with an exponentially decaying average of the squared gradients in the denominator of the update rule. The value assigned must be in the range (0,1).

conditioningConst

Specifies a conditioning constant for the ADADELTA adaptive optimization algorithm that is used to condition the step size in regions where the exponentially decaying average of the squared gradients is small. The value assigned must be in the range (0,1).

learningRate

Specifies the size of the step taken in the direction of the negative gradient for each iteration of the learning process. The default value is `= 0.001`.

momentum

Specifies weights for each dimension that control the contribution of the previous step to the size of the next step during training. This modifies the `learningRate` to speed up training. The value must be `>= 0` and `< 1`.

`nag`

If `TRUE`, Nesterov's Accelerated Gradient Descent is used. This method reduces the oracle complexity of gradient descent and is optimal for smooth convex optimization.

`weightDecay`

Specifies the scaling weights for the step size. After each weight update, the weights in the network are scaled by $(1 - \text{learningRate} * \text{weightDecay})$. The value must be `>= 0` and `< 1`.

`lRateRedRatio`

Specifies the learning rate reduction ratio: the ratio by which the learning rate is reduced during training. Reducing the learning rate can avoid local minima. The value must be `> 0` and `<= 1`.

- A value of `1.0` means no reduction.
- A value of `0.9` means the learning rate is reduced to 90 its current value. The reduction can be triggered either periodically, to occur after a fixed number of iterations, or when a certain error criteria concerning increases or decreases in the loss function are satisfied.
- To trigger a periodic rate reduction, specify the frequency by setting the number of iterations between reductions with the `lRateRedFreq` argument.
- To trigger rate reduction based on an error criterion, specify a number in `lRateRedErrorRatio`.

`lRateRedFreq`

Sets the learning rate reduction frequency by specifying number of iterations between reductions. For example, if `10` is specified, the learning rate is reduced once every 10 iterations.

`lRateRedErrorRatio`

Specifies the learning rate reduction error criterion. If set to `0`, the learning rate is reduced if the loss increases between iterations. If set to a fractional value greater than `0`, the learning rate is reduced if the loss decreases by less than that fraction of its previous value.

Details

These functions can be used for the `optimizer` argument in `rxNeuralNet`.

The `sgd` function specifies Stochastic Gradient Descent. `maOptimizer`

The `adaDeltaSgd` function specifies the AdaDelta gradient descent, described in the 2012 paper "ADADELTA: An Adaptive Learning Rate Method" by Matthew D. Zeiler.

Value

A character string that contains the specification for the optimization algorithm.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#)

References

[ADADELTA: An Adaptive Learning Rate Method](#)

See also

`rxNeuralNet`,

Examples

```
myIris = iris
myIris$Setosa <- iris$Species == "setosa"

res1 <- rxNeuralNet(formula = Setosa~Sepal.Length + Sepal.Width +
  Petal.Width,
  data = myIris,
```

```
optimizer = sgd(learningRate = .002))

res2 <- rxNeuralNet(formula = Setosa~Sepal.Length + Sepal.Width +
Petal.Width,
  data = myIris,
  optimizer = adaDeltaSgd(decay = .9, conditioningConst = 1e-05))
```

resizeImage: Machine Learning Resize Image Transform

Article • 02/28/2023

Resizes an image to a specified dimension using a specified resizing method.

Usage

```
resizeImage(vars, width = 224, height = 224, resizingOption = "IsoCrop")
```

Arguments

vars

A named list of character vectors of input variable names and the name of the output variable. Note that the input variables must be of the same type. For one-to-one mappings between input and output variables, a named character vector can be used.

width

Specifies the width of the scaled image in pixels. The default value is 224.

height

Specifies the height of the scaled image in pixels. The default value is 224.

resizingOption

Specifies the resizing method to use. Note that all methods are using bilinear interpolation. The options are:

- **"IsoPad"**: The image is resized such that the aspect ratio is preserved. If needed, the image is padded with black to fit the new width or height.

- `"IsoCrop"`: The image is resized such that the aspect ratio is preserved. If needed, the image is cropped to fit the new width or height.
- `"Aniso"`: The image is stretched to the new width and height, without preserving the aspect ratio. The default value is `"IsoPad"`.

Details

`resizeImage` resizes an image to the specified height and width using a specified resizing method. The input variables to this transform must be images, typically the result of the `loadImage` transform.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

Examples

```
train <- data.frame(Path =
c(system.file("help/figures/RevolutionAnalyticslogo.png", package =
"MicrosoftML")), Label = c(TRUE), stringsAsFactors = FALSE)

# Loads the images from variable Path, resizes the images to 1x1 pixels and
trains a neural net.
model <- rxNeuralNet(
  Label ~ Features,
  data = train,
  mlTransforms = list(
    loadImage(vars = list(Features = "Path")),
    resizeImage(vars = "Features", width = 1, height = 1, resizing =
"Aniso"),
    extractPixels(vars = "Features")
  ),
  mlTransformVars = "Path",
  numHiddenNodes = 1,
  numIterations = 1)

# Featurizes the images from variable Path using the default model, and
```

trains a linear model on the result.

```
model <- rxFastLinear(  
  Label ~ Features,  
  data = train,  
  mlTransforms = list(  
    loadImage(vars = list(Features = "Path")),  
    resizeImage(vars = "Features", width = 224, height = 224), # If  
dnnModel == "AlexNet", the image has to be resized to 227x227.  
    extractPixels(vars = "Features"),  
    featurizeImage(var = "Features")  
  ),  
  mlTransformVars = "Path")
```

rxEnsemble: Ensembles

Article • 02/28/2023

Train an ensemble of models

Usage

```
rxEnsemble(formula = NULL, data, trainers, type = c("binary",
"regression",
"multiClass", "anomaly"), randomSeed = NULL,
modelCount = length(trainers), replace = FALSE, sampRate = NULL,
splitData = FALSE, combineMethod = c("median", "average", "vote"),
maxCalibration = 1e+05, mlTransforms = NULL, mlTransformVars = NULL,
rowSelection = NULL, transforms = NULL, transformObjects = NULL,
transformFunc = NULL, transformVars = NULL, transformPackages = NULL,
transformEnvir = NULL, blocksPerRead = rxGetOption("blocksPerRead"),
reportProgress = rxGetOption("reportProgress"), verbose = 1,
computeContext = rxGetOption("computeContext"), ...)
```

Arguments

formula

The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.

data

A data source object or a character string specifying a `.xdffile` or a data frame object. Alternatively, it can be a list of data sources indicating each model should be trained using one of the data sources in the list. In this case, the length of the data list must be equal to `modelCount`.

trainers

A list of trainers with their arguments. The trainers are created by using [fastTrees](#), [fastForest](#), [fastLinear](#), [logisticRegression](#) or [neuralNet](#).

type

A character string that specifies the type of ensemble: "binary" for Binary Classification or "regression" for Regression.

randomSeed

Specifies the random seed. The default value is `NULL`.

modelCount

Specifies the number of models to train. If this number is greater than the length of the trainers list, the trainers list is duplicated to match `modelCount`.

replace

A logical value specifying if the sampling of observations should be done with or without replacement. The default value is `FALSE`.

sampRate

a scalar of positive value specifying the percentage of observations to sample for each trainer. The default is 1.0 for sampling with replacement (i.e., `replace=TRUE`) and 0.632 for sampling without replacement (i.e., `replace=FALSE`). When `splitData` is `TRUE`, the default of `sampRate` is 1.0 (no sampling is done before splitting).

splitData

A logical value specifying whether or not to train the base models on non-overlapping partitions. The default is `FALSE`. It is available only for `RxSpark` compute context and ignored for others.

combineMethod

Specifies the method used to combine the models:

- `median` to compute the median of the individual model outputs,
- `average` to compute the average of the individual model outputs and
- `vote` to compute (pos-neg) / the total number of models, where 'pos' is the number of positive outputs and 'neg' is the number of negative outputs.

maxCalibration

Specifies the maximum number of examples to use for calibration. This argument is ignored for all tasks other than binary classification.

m1Transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. Transforms that require an additional pass over the data (such as `featurizeText`, `categorical`) are not allowed. These transformations are performed after any specified R transformations. The default value is `NULL`.

m1TransformVars

Specifies a character vector of variable names to be used in `m1Transforms` or `NULL` if none are to be used. The default value is `NULL`.

rowSelection

Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

transforms

An expression of the form `list(name = expression, ``...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function. The default value is `NULL`.

transformObjects

A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`. The default value is `NULL`.

`transformFunc`

The variable transformation function. See `rxTransform` for details. The default value is `NULL`.

`transformVars`

A character vector of input data set variables needed for the transformation function. See `rxTransform` for details. The default value is `NULL`.

`transformPackages`

A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded. The default value is `NULL`.

`transformEnvir`

A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead. The default value is `NULL`.

`blocksPerRead`

Specifies the number of blocks to read for each chunk of data read from the data source.

`reportProgress`

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.

- **1**: the number of processed rows is printed and updated.
- **2**: rows processed and timings are reported.
- **3**: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If **0**, no verbose output is printed during calculations. Integer values from **1** to **4** provide increasing amounts of information. The default value is **1**.

computeContext

Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxSpark` compute contexts are supported. When `RxSpark` is specified, the training of the models is done in a distributed way, and the ensembling is done locally. Note that the compute context cannot be non-waiting.

...

Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

`/codexEnsemble` is a function that trains a number of models of various kinds to obtain better predictive performance than could be obtained from a single model.

Value

A `rxEnsemble` object with the trained ensemble model.

Examples

```
# Create an ensemble of regression rxFastTrees models

# use xdf data source
dataFile <- file.path(rxGetOption("sampleDataDir"), "claims4blocks.xdf")
rxGetInfo(dataFile, getVarInfo = TRUE, getBlockSizes = TRUE)
form <- cost ~ age + type + number
```

```

rxSetComputeContext("localpar")
rxGetComputeContext()

# build an ensemble model that contains three 'rxFastTrees' models with
different parameters
ensemble <- rxEnsemble(
  formula = form,
  data = dataFile,
  type = "regression",
  trainers = list(fastTrees(), fastTrees(numTrees = 60),
fastTrees(learningRate = 0.1)), #a list of trainers with their arguments.
  replace = TRUE # Indicates using a bootstrap sample for each trainer
)

# use text data source
colInfo <- list(DayOfWeek = list(type = "factor", levels = c("Monday",
"Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")))

source <- system.file("SampleData/AirlineDemoSmall.csv", package =
"RevoScaleR")
data <- RxTextData(source, missingValueString = "M", colInfo = colInfo)

# When 'distributed' is TRUE distributed data source is created
distributed <- FALSE
if (distributed) {
  bigDataDirRoot <- "/share"
  inputDir <- file.path(bigDataDirRoot, "AirlineDemoSmall")
  rxHadoopMakeDir(inputDir)
  rxHadoopCopyFromLocal(source, inputDir)
  hdfsFS <- RxHdfsFileSystem()
  data <- RxTextData(file = inputDir, missingValueString = "M", colInfo =
colInfo, fileSystem = hdfsFS)
}

# When 'distributed' is TRUE training is distributed
if (distributed) {
  cc <- rxSetComputeContext(RxSpark())
} else {
  cc <- rxGetComputeContext()
}

ensemble <- rxEnsemble(
  formula = ArrDelay ~ DayOfWeek,
  data = data,
  type = "regression",
  trainers = list(fastTrees(), fastTrees(numTrees = 60),
fastTrees(learningRate = 0.1)), # The ensemble will contain three
'rxFastTrees' models
  replace = TRUE # Indicates using a bootstrap sample for each trainer
)

# Change the compute context back to previous for scoring
rxSetComputeContext(cc)

```



```
# Put score and model variables in data frame
scores <- rxPredict(ensemble, data = data, writeModelVars = TRUE)

# Plot actual versus predicted values with smoothed line
rxLinePlot(Score ~ ArrDelay, type = c("p", "smooth"), data = scores)
```

rxFastForest: Fast Forest

Article • 02/28/2023

Machine Learning Fast Forest

Usage

```
rxFastForest(formula = NULL, data, type = c("binary", "regression"),
  numTrees = 100, numLeaves = 20, minSplit = 10, exampleFraction = 0.7,
  featureFraction = 0.7, splitFraction = 0.7, numBins = 255,
  firstUsePenalty = 0, gainConfLevel = 0, trainThreads = 8,
  randomSeed = NULL, mlTransforms = NULL, mlTransformVars = NULL,
  rowSelection = NULL, transforms = NULL, transformObjects = NULL,
  transformFunc = NULL, transformVars = NULL, transformPackages = NULL,
  transformEnvir = NULL, blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 2,
  computeContext = rxGetOption("computeContext"),
  ensemble = ensembleControl(), ...)
```

Arguments

formula

The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.

data

A data source object or a character string specifying a `.xdf` file or a data frame object.

type

A character string denoting Fast Tree type:

- `"binary"` for the default Fast Tree Binary Classification or
- `"regression"` for Fast Tree Regression.

numTrees

Specifies the total number of decision trees to create in the ensemble. By creating more decision trees, you can potentially get better coverage, but the training time increases. The default value is 100.

numLeaves

The maximum number of leaves (terminal nodes) that can be created in any tree. Higher values potentially increase the size of the tree and get better precision, but risk overfitting and requiring longer training times. The default value is 20.

minSplit

Minimum number of training instances required to form a leaf. That is, the minimal number of documents allowed in a leaf of a regression tree, out of the sub-sampled data. A 'split' means that features in each level of the tree (node) are randomly divided. The default value is 10.

exampleFraction

The fraction of randomly chosen instances to use for each tree. The default value is 0.7.

featureFraction

The fraction of randomly chosen features to use for each tree. The default value is 0.7.

splitFraction

The fraction of randomly chosen features to use on each split. The default value is 0.7.

numBins

Maximum number of distinct values (bins) per feature. The default value is 255.

firstUsePenalty

The feature first use penalty coefficient. The default value is 0.

gainConfLevel

Tree fitting gain confidence requirement (should be in the range [0,1)). The default value is 0.

trainThreads

The number of threads to use in training. If `NULL` is specified, the number of threads to use is determined internally. The default value is `NULL`.

randomSeed

Specifies the random seed. The default value is `NULL`.

m1Transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See [featurizeText](#), [categorical](#), and [categoricalHash](#), for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.

m1TransformVars

Specifies a character vector of variable names to be used in `m1Transforms` or `NULL` if none are to be used. The default value is `NULL`.

rowSelection

Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

transforms

An expression of the form `list(name = expression, ``...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

transformObjects

A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

The variable transformation function. See `rxTransform` for details.

transformVars

A character vector of input data set variables needed for the transformation function. See `rxTransform` for details.

transformPackages

A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.

transformEnvir

A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

blocksPerRead

Specifies the number of blocks to read for each chunk of data read from the data source.

reportProgress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

computeContext

Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently `local` and `RxInSqlServer` compute contexts are supported.

ensemble

Control parameters for ensembling.



Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

Decision trees are non-parametric models that perform a sequence of simple tests on inputs. This decision procedure maps them to outputs found in the training dataset whose inputs were similar to the instance being processed. A decision is made at each node of the binary tree data structure based on a measure of similarity that maps each instance recursively through the branches of the tree until the appropriate leaf node is reached and the output decision returned.

Decision trees have several advantages:

They are efficient in both computation and memory usage during training and prediction.

They can represent non-linear decision boundaries.

They perform integrated feature selection and classification.

They are resilient in the presence of noisy features.

Fast forest regression is a random forest and quantile regression forest implementation using the regression tree learner in [rxFastTrees](#). The model consists of an ensemble of decision trees. Each tree in a decision forest outputs a Gaussian distribution by way of prediction. An aggregation is performed over the ensemble of trees to find a Gaussian distribution closest to the combined distribution for all trees in the model.

This decision forest classifier consists of an ensemble of decision trees. Generally, ensemble models provide better coverage and accuracy than single decision trees. Each tree in a decision forest outputs a Gaussian distribution by way of prediction. An aggregation is performed over the ensemble of trees to find a Gaussian distribution closest to the combined distribution for all trees in the model.

Value

`rxFastForest`: A `rxFastForest` object with the trained model.

`FastForest`: A learner specification object of class `maml` for the Fast Forest trainer.

Notes

This algorithm is multi-threaded and will always attempt to load the entire dataset into memory.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) [↗](#)

References

[Wikipedia: Random forest](#) [↗](#)

See also

[rxFastTrees](#), [rxFastLinear](#), [rxLogisticRegression](#), [rxNeuralNet](#), [rxOneClassSvm](#), [featurizeText](#), [categorical](#), [categoricalHash](#), [rxPredict.mlModel](#).

Examples

```
# Estimate a binary classification forest
infert1 <- infert
infert1$isCase = (infert1$case == 1)
forestModel <- rxFastForest(formula = isCase ~ age + parity + education +
spontaneous + induced,
    data = infert1)

# Create text file with per-instance results using rxPredict
txtOutFile <- tempfile(pattern = "scoreOut", fileext = ".txt")
txtOutDS <- RxTextData(file = txtOutFile)
scoreDS <- rxPredict(forestModel, data = infert1,
    extraVarsToWrite = c("isCase", "Score"), outData = txtOutDS)

# Print the first ten rows
rxDataStep(scoreDS, numRows = 10)

# Clean-up
file.remove(txtOutFile)

#####
# Estimate a regression fast forest

# Use the built-in data set 'airquality' to create test and train data
DF <- airquality[!is.na(airquality$Ozone), ]
DF$Ozone <- as.numeric(DF$Ozone)
randomSplit <- rnorm(nrow(DF))
trainAir <- DF[randomSplit >= 0,]
testAir <- DF[randomSplit < 0,]
airFormula <- Ozone ~ Solar.R + Wind + Temp

# Regression Fast Forest for train data
rxFastForestReg <- rxFastForest(airFormula, type = "regression",
    data = trainAir)

# Put score and model variables in data frame
rxFastForestScoreDF <- rxPredict(rxFastForestReg, data = testAir,
```



```
writeModelVars = TRUE)  
  
# Plot actual versus predicted values with smoothed line  
rxLinePlot(Score ~ Ozone, type = c("p", "smooth"), data =  
rxFastForestScoreDF)
```

rxFastLinear: Fast Linear Model -- Stochastic Dual Coordinate Ascent

Article • 02/28/2023

A Stochastic Dual Coordinate Ascent (SDCA) optimization trainer for linear binary classification and regression.

`rxFastLinear` is a trainer based on the Stochastic Dual Coordinate Ascent (SDCA) method, a state-of-the-art optimization technique for convex objective functions. The algorithm can be scaled for use on large out-of-memory data sets due to a semi-asynchronized implementation that supports multi-threading. primal and dual updates in a separate thread. Several choices of loss functions are also provided. The SDCA method combines several of the best properties and capabilities of logistic regression and SVM algorithms. For more information on SDCA, see the citations in the reference section.

Traditional optimization algorithms, such as stochastic gradient descent (SGD), optimize the empirical loss function directly. The SDCA chooses a different approach that optimizes the dual problem instead. The dual loss function is parametrized by per-example weights. In each iteration, when a training example from the training data set is read, the corresponding example weight is adjusted so that the dual loss function is optimized with respect to the current example. No learning rate is needed by SDCA to determine step size as is required by various gradient descent methods.

`rxFastLinear` supports binary classification with three types of loss functions currently: Log loss, hinge loss, and smoothed hinge loss. Linear regression also supports with squared loss function. Elastic net regularization can be specified by the `l2Weight` and `l1Weight` parameters. Note that the `l2Weight` has an effect on the rate of convergence. In general, the larger the `l2Weight`, the faster SDCA converges.

Note that `rxFastLinear` is a stochastic and streaming optimization algorithm. The result depends on the order of the training data. For reproducible results, it is recommended that one sets `shuffle` to `FALSE` and `trainThreads` to `1`.

Usage

```
rxFastLinear(formula = NULL, data, type = c("binary", "regression"),  
             lossFunction = NULL, l2Weight = NULL, l1Weight = NULL,
```

```
trainThreads = NULL, convergenceTolerance = 0.1, maxIterations = NULL,
shuffle = TRUE, checkFrequency = NULL, normalize = "auto",
mlTransforms = NULL, mlTransformVars = NULL, rowSelection = NULL,
transforms = NULL, transformObjects = NULL, transformFunc = NULL,
transformVars = NULL, transformPackages = NULL, transformEnvir = NULL,
blocksPerRead = rxGetOption("blocksPerRead"),
reportProgress = rxGetOption("reportProgress"), verbose = 1,
computeContext = rxGetOption("computeContext"),
ensemble = ensembleControl(), ...)
```

Arguments

formula

The formula described in `rxFormula`. Interaction terms and `F()` are currently not supported in **MicrosoftML**.

data

A data source object or a character string specifying a `.xdf` file or a data frame object.

type

Specifies the model type with a character string: `"binary"` for the default binary classification or `"regression"` for linear regression.

lossFunction

Specifies the empirical loss function to optimize. For binary classification, the following choices are available:

- `logLoss`: The log-loss. This is the default.
- `hingeLoss`: The SVM hinge loss. Its parameter represents the margin size.
- `smoothHingeLoss`: The smoothed hinge loss. Its parameter represents the smoothing constant.

For linear regression, squared loss `squaredLoss` is currently supported. When this parameter is set to `NULL`, its default value depends on the type of learning:

- `logLoss` for binary classification.
- `squaredLoss` for linear regression.

l2Weight

Specifies the L2 regularization weight. The value must be either non-negative or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. `NULL` is the default value.

l1Weight

Specifies the L1 regularization weight. The value must be either non-negative or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. `NULL` is the default value.

trainThreads

Specifies how many concurrent threads can be used to run the algorithm. When this parameter is set to `NULL`, the number of threads used is determined based on the number of logical processors available to the process as well as the sparsity of data. Set it to `1` to run the algorithm in a single thread.

convergenceTolerance

Specifies the tolerance threshold used as a convergence criterion. It must be between 0 and 1. The default value is `0.1`. The algorithm is considered to have converged if the relative duality gap, which is the ratio between the duality gap and the primal loss, falls below the specified convergence tolerance.

maxIterations

Specifies an upper bound on the number of training iterations. This parameter must be positive or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. Each iteration requires a complete pass over the training data. Training terminates after the total number of iterations reaches the specified upper bound or when the loss function converges, whichever happens earlier.

shuffle

Specifies whether to shuffle the training data. Set `TRUE` to shuffle the data; `FALSE` not to shuffle. The default value is `TRUE`. SDCA is a stochastic optimization algorithm. If shuffling is turned on, the training data is shuffled on each iteration.

checkFrequency

The number of iterations after which the loss function is computed and checked to determine whether it has converged. The value specified must be a positive integer or `NULL`. If `NULL`, the actual value is automatically computed based on data set. Otherwise, for example, if `checkFrequency = 5` is specified, then the loss function is computed and convergence is checked every 5 iterations. The computation of the loss function requires a separate complete pass over the training data.

normalize

Specifies the type of automatic normalization used:

- `"auto"`: if normalization is needed, it is automatically performed. This is the default value.
- `"no"`: no normalization is performed.
- `"yes"`: normalization is performed.
- `"warn"`: if normalization is needed, a warning message is displayed, but normalization is not performed.

Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MinMax` normalizer is used. It normalizes values in an interval $[a, b]$ where $-1 \leq a \leq 0$ and $0 \leq b \leq 1$ and $b - a = 1$. This normalizer preserves sparsity by mapping zero to zero.

m1Transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See [featurizeText](#), [categorical](#), and [categoricalHash](#), for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.

m1TransformVars

Specifies a character vector of variable names to be used in `m1Transforms` or `NULL` if none are to be used. The default value is `NULL`.

rowSelection

Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

transforms

An expression of the form `list(name = expression, ``...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

transformObjects

A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

The variable transformation function. See `rxTransform` for details.

transformVars

A character vector of input data set variables needed for the transformation function. See `rxTransform` for details.

transformPackages

A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in `RevoScaleR` functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may

also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.

transformEnvir

A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

blocksPerRead

Specifies the number of blocks to read for each chunk of data read from the data source.

reportProgress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

computeContext

Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently `local` and `RxInSqlServer` compute contexts are supported.

ensemble

Control parameters for ensembling.

...

Additional arguments to be passed directly to the Microsoft Compute Engine.

Value

`rxFastLinear`: A `rxFastLinear` object with the trained model.

`FastLinear`: A learner specification object of class `maml` for the Fast Linear trainer.

Notes

This algorithm is multi-threaded and will not attempt to load the entire dataset into memory.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) ↗

References

[Scaling Up Stochastic Dual Coordinate Ascent](#) ↗

[Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization](#) ↗

See also

[logLoss](#), [hingeLoss](#), [smoothHingeLoss](#), [squaredLoss](#), [rxFastTrees](#), [rxFastForest](#), [rxLogisticRegression](#), [rxNeuralNet](#), [rxOneClassSvm](#), [featurizeText](#), [categorical](#), [categoricalHash](#), [rxPredict.mlModel](#).

Examples

```
# Train a binary classification model with rxFastLinear
res1 <- rxFastLinear(isCase ~ age + parity + education + spontaneous +
  induced,
                    transforms = list(isCase = case == 1),
                    data = infert,
                    type = "binary")
# Print a summary of the model
```



```

summary(res1)

# Score to a data frame
scoreDF <- rxPredict(res1, data = infert,
  extraVarsToWrite = "isCase")

# Compute and plot the Radio Operator Curve and AUC
roc1 <- rxRoc(actualVarName = "isCase", predVarNames = "Probability", data
= scoreDF)
plot(roc1)
rxAuc(roc1)

#####
# rxFastLinear Regression

# Create an xdf file with the attitude data
myXdf <- tempfile(pattern = "tempAttitude", fileext = ".xdf")
rxDataStep(attitude, myXdf, rowsPerRead = 50, overwrite = TRUE)
myXdfDS <- RxXdfData(file = myXdf)

attitudeForm <- rating ~ complaints + privileges + learning +
  raises + critical + advance

# Estimate a regression model with rxFastLinear
res2 <- rxFastLinear(formula = attitudeForm, data = myXdfDS,
  type = "regression")

# Score to data frame
scoreOut2 <- rxPredict(res2, data = myXdfDS,
  extraVarsToWrite = "rating")

# Plot the rating versus the score with a regression line
rxLinePlot(rating~Score, type = c("p","r"), data = scoreOut2)

# Clean up
file.remove(myXdf)

```

rxFastTrees: Fast Tree

Article • 02/28/2023

Machine Learning Fast Tree

Usage

```
rxFastTrees(formula = NULL, data, type = c("binary", "regression"),
  numTrees = 100, numLeaves = 20, learningRate = 0.2, minSplit = 10,
  exampleFraction = 0.7, featureFraction = 1, splitFraction = 1,
  numBins = 255, firstUsePenalty = 0, gainConfLevel = 0,
  unbalancedSets = FALSE, trainThreads = 8, randomSeed = NULL,
  mlTransforms = NULL, mlTransformVars = NULL, rowSelection = NULL,
  transforms = NULL, transformObjects = NULL, transformFunc = NULL,
  transformVars = NULL, transformPackages = NULL, transformEnvir = NULL,
  blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 2,
  computeContext = rxGetOption("computeContext"),
  ensemble = ensembleControl(), ...)
```

Arguments

formula

The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.

data

A data source object or a character string specifying a `.xdf` file or a data frame object.

type

A character string that specifies the type of Fast Tree: `"binary"` for the default Fast Tree Binary Classification or `"regression"` for Fast Tree Regression.

numTrees

Specifies the total number of decision trees to create in the ensemble. By creating more decision trees, you can potentially get better coverage, but the training time increases. The default value is 100.

numLeaves

The maximum number of leaves (terminal nodes) that can be created in any tree. Higher values potentially increase the size of the tree and get better precision, but risk overfitting and requiring longer training times. The default value is 20.

learningRate

Determines the size of the step taken in the direction of the gradient in each step of the learning process. This determines how fast or slow the learner converges on the optimal solution. If the step size is too big, you might overshoot the optimal solution. If the step size is too small, training takes longer to converge to the best solution.

minSplit

Minimum number of training instances required to form a leaf. That is, the minimal number of documents allowed in a leaf of a regression tree, out of the sub-sampled data. A 'split' means that features in each level of the tree (node) are randomly divided. The default value is 10. Only the number of instances is counted even if instances are weighted.

exampleFraction

The fraction of randomly chosen instances to use for each tree. The default value is 0.7.

featureFraction

The fraction of randomly chosen features to use for each tree. The default value is 1.

splitFraction

The fraction of randomly chosen features to use on each split. The default value is 1.

numBins

Maximum number of distinct values (bins) per feature. If the feature has fewer values than the number indicated, each value is placed in its own bin. If there are more values, the algorithm creates `numBins` bins.

firstUsePenalty

The feature first use penalty coefficient. This is a form of regularization that incurs a penalty for using a new feature when creating the tree. Increase this value to create trees that don't use many features. The default value is 0.

gainConfLevel

Tree fitting gain confidence requirement (should be in the range [0,1)). The default value is 0.

unbalancedSets

If `TRUE`, derivatives optimized for unbalanced sets are used. Only applicable when `type` equal to `"binary"`. The default value is `FALSE`.

trainThreads

The number of threads to use in training. The default value is 8.

randomSeed

Specifies the random seed. The default value is `NULL`.

m1Transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See [featurizeText](#), [categorical](#), and [categoricalHash](#), for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.

m1TransformVars

Specifies a character vector of variable names to be used in `m1Transforms` or `NULL` if none are to be used. The default value is `NULL`.

rowSelection

Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

transforms

An expression of the form `list(name = expression, ``...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

transformObjects

A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

The variable transformation function. See `rxTransform` for details.

transformVars

A character vector of input data set variables needed for the transformation function. See `rxTransform` for details.

transformPackages

A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in `RevoScaleR` functions via their `transforms` and `transformFunc` arguments or those defined implicitly

via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.

transformEnvir

A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

blocksPerRead

Specifies the number of blocks to read for each chunk of data read from the data source.

reportProgress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

computeContext

Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently `local` and `RxInSqlServer` compute contexts are supported.

ensemble

Control parameters for ensembling.



Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

rxFastTrees is an implementation of FastRank. FastRank is an efficient implementation of the MART gradient boosting algorithm. Gradient boosting is a machine learning technique for regression problems. It builds each regression tree in a step-wise fashion, using a predefined loss function to measure the error for each step and corrects for it in the next. So this prediction model is actually an ensemble of weaker prediction models. In regression problems, boosting builds a series of such trees in a step-wise fashion and then selects the optimal tree using an arbitrary differentiable loss function.

MART learns an ensemble of regression trees, which is a decision tree with scalar values in its leaves. A decision (or regression) tree is a binary tree-like flow chart, where at each interior node one decides which of the two child nodes to continue to based on one of the feature values from the input. At each leaf node, a value is returned. In the interior nodes, the decision is based on the test " $x \leq v$ ", where x is the value of the feature in the input sample and v is one of the possible values of this feature. The functions that can be produced by a regression tree are all the piece-wise constant functions.

The ensemble of trees is produced by computing, in each step, a regression tree that approximates the gradient of the loss function, and adding it to the previous tree with coefficients that minimize the loss of the new tree. The output of the ensemble produced by MART on a given instance is the sum of the tree outputs.

In case of a binary classification problem, the output is converted to a probability by using some form of calibration.

In case of a regression problem, the output is the predicted value of the function.

In case of a ranking problem, the instances are ordered by the output value of the ensemble.

If `type` is set to `"regression"`, a regression version of FastTree is used. If set to `"ranking"`, a ranking version of FastTree is used. In the ranking case, the instances should be ordered by the output of the tree ensemble. The only difference in the settings of these versions is in the calibration settings, which are needed only for classification.

Value

`rxFastTrees`: A `rxFastTrees` object with the trained model.

`FastTree`: A learner specification object of class `mam1` for the Fast Tree trainer.

Notes

This algorithm is multi-threaded and will always attempt to load the entire dataset into memory.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) ↗

References

[Wikipedia: Gradient boosting \(Gradient tree boosting\)](#) ↗

[Greedy function approximation: A gradient boosting machine.](#) ↗

See also

[rxFastForest](#), [rxFastLinear](#), [rxLogisticRegression](#), [rxNeuralNet](#), [rxOneClassSvm](#), [featurizeText](#), [categorical](#), [categoricalHash](#), [rxPredict.mlModel](#).

Examples

```
# Estimate a binary classification tree
infert1 <- infert
infert1$isCase = (infert1$case == 1)
treeModel <- rxFastTrees(formula = isCase ~ age + parity + education +
  spontaneous + induced,
  data = infert1)

# Create xdf file with per-instance results using rxPredict
xdfOut <- tempfile(pattern = "scoreOut", fileext = ".xdf")
scoreDS <- rxPredict(treeModel, data = infert1,
  extraVarsToWrite = c("isCase", "Score"),
  outData = xdfOut)
```



```

rxDataStep(scoreDS, numRows = 10)

# Clean-up
file.remove(xdfOut)

#####
# Estimate a regression fast tree

# Use the built-in data set 'airquality' to create test and train data
DF <- airquality[!is.na(airquality$Ozone), ]
DF$Ozone <- as.numeric(DF$Ozone)
randomSplit <- rnorm(nrow(DF))
trainAir <- DF[randomSplit >= 0,]
testAir <- DF[randomSplit < 0,]
airFormula <- Ozone ~ Solar.R + Wind + Temp

# Regression Fast Tree for train data
fastTreeReg <- rxFastTrees(airFormula, type = "regression",
  data = trainAir)

# Put score and model variables in data frame
fastTreeScoreDF <- rxPredict(fastTreeReg, data = testAir,
  writeModelVars = TRUE)

# Plot actual versus predicted values with smoothed line
rxLinePlot(Score ~ Ozone, type = c("p", "smooth"), data = fastTreeScoreDF)

```

rxFeaturize: Data Transformation for RevoScaleR data sources

Article • 02/28/2023

Transforms data from an input data set to an output data set.

Usage

```
rxFeaturize(data, outData = NULL, overwrite = FALSE, dataThreads = NULL,
  randomSeed = NULL, maxSlots = 5000, mlTransforms = NULL,
  mlTransformVars = NULL, rowSelection = NULL, transforms = NULL,
  transformObjects = NULL, transformFunc = NULL, transformVars = NULL,
  transformPackages = NULL, transformEnvir = NULL,
  blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 1,
  computeContext = rxGetOption("computeContext"), ...)
```

Arguments

data

A RevoScaleR data source object, a data frame, or the path to a `.xdf` file.

outData

Output text or xdf file name or an `RxDataSource` with write capabilities in which to store transformed data. If `NULL`, a data frame is returned. The default value is `NULL`.

overwrite

If `TRUE`, an existing `outData` is overwritten; if `FALSE` an existing `outData` is not overwritten. The default value is `FALSE`.

dataThreads

An integer specifying the desired degree of parallelism in the data pipeline. If `NULL`, the number of threads used is determined internally. The default value is `NULL`.

`randomSeed`

Specifies the random seed. The default value is `NULL`.

`maxSlots`

Max slots to return for vector valued columns (`<=0` to return all).

`m1Transforms`

Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See [featurizeText](#), [categorical](#), and [categoricalHash](#), for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.

`m1TransformVars`

Specifies a character vector of variable names to be used in `m1Transforms` or `NULL` if none are to be used. The default value is `NULL`.

`rowSelection`

Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

An expression of the form `list(name = expression, ``...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function. The default value is `NULL`.

`transformObjects`

A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`. The default value is `NULL`.

`transformFunc`

The variable transformation function. See `rxTransform` for details. The default value is `NULL`.

`transformVars`

A character vector of input data set variables needed for the transformation function. See `rxTransform` for details. The default value is `NULL`.

`transformPackages`

A character vector specifying additional R packages (outside of those specified in `rxgetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxgetOption("transformPackages")` are preloaded. The default value is `NULL`.

`transformEnvir`

A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead. The default value is `NULL`.

`blocksPerRead`

Specifies the number of blocks to read for each chunk of data read from the data source.

reportProgress

An integer value that specifies the level of reporting on the row processing progress:

- 0: no progress is reported.
- 1: the number of processed rows is printed and updated.
- 2: rows processed and timings are reported.
- 3: rows processed and all timings are reported.

The default value is 1.

verbose

An integer value that specifies the amount of output wanted. If 0, no verbose output is printed during calculations. Integer values from 1 to 4 provide increasing amounts of information. The default value is 1.

computeContext

Sets the context in which computations are executed, specified with a valid RxComputeContext. Currently local and RxInSqlServer compute contexts are supported.



Additional arguments to be passed directly to the Microsoft Compute Engine.

Value

A data frame or an RxDataSource object representing the created output data.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

See also

rxDataStep, rxImport, rxTransform.

Examples

```
# rxFeaturize basically allows you to access data from the MicrosoftML
transforms
# In this example we'll look at getting the output of the categorical
transform

# Create the data
categoricalData <- data.frame(
  placesVisited = c(
    "London",
    "Brunei",
    "London",
    "Paris",
    "Seria"
  ),
  stringsAsFactors = FALSE
)

# Invoke the categorical transform
categorized <- rxFeaturize(
  data = categoricalData,
  mlTransforms = list(categorical(vars = c(xDataCat = "placesVisited")))
)

# Now let's look at the data
categorized
```

rxHashEnv: An environment object used to store package-wide state.

Article • 02/28/2023

An environment object used to store package-wide state.

Usage

```
rxHashEnv
```

Format

An object of class `environment` of length 2.

rxLogisticRegression: Logistic Regression

Article • 02/28/2023

Machine Learning Logistic Regression

Usage

```
rxLogisticRegression(formula = NULL, data, type = c("binary",
"multiClass"),
  l2Weight = 1, l1Weight = 1, optTol = 1e-07, memorySize = 20,
  initWtsScale = 0, maxIterations = 2147483647, showTrainingStats = FALSE,
  sgdInitTol = 0, trainThreads = NULL, denseOptimizer = FALSE,
  normalize = "auto", mlTransforms = NULL, mlTransformVars = NULL,
  rowSelection = NULL, transforms = NULL, transformObjects = NULL,
  transformFunc = NULL, transformVars = NULL, transformPackages = NULL,
  transformEnvir = NULL, blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 1,
  computeContext = rxGetOption("computeContext"),
  ensemble = ensembleControl(), ...)
```

Arguments

formula

The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the `MicrosoftML`.

data

A data source object or a character string specifying a `.xdf` file or a data frame object.

type

A character string that specifies the type of Logistic Regression: `"binary"` for the default binary classification logistic regression or `"multi"` for multinomial logistic regression.

l2Weight

The L2 regularization weight. Its value must be greater than or equal to 0 and the default value is set to 1 .

l1Weight

The L1 regularization weight. Its value must be greater than or equal to 0 and the default value is set to 1 .

optTol

Threshold value for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model. Smaller values are slower, but more accurate. The default value is $1e-07$.

memorySize

Memory size for L-BFGS, specifying the number of past positions and gradients to store for the computation of the next step. This optimization parameter limits the amount of memory that is used to compute the magnitude and direction of the next step. When you specify less memory, training is faster but less accurate. Must be greater than or equal to 1 and the default value is 20 .

initWtsScale

Sets the initial weights diameter that specifies the range from which values are drawn for the initial weights. These weights are initialized randomly from within this range. For example, if the diameter is specified to be d , then the weights are uniformly distributed between $-d/2$ and $d/2$. The default value is 0 , which specifies that all the weights are initialized to 0 .

maxIterations

Sets the maximum number of iterations. After this number of steps, the algorithm stops even if it has not satisfied convergence criteria.

showTrainingStats

Specify `TRUE` to show the statistics of training data and the trained model; otherwise, `FALSE`. The default value is `FALSE`. For additional information about model statistics, see [summary.mlModel](#).

`sgdInitTol`

Set to a number greater than 0 to use Stochastic Gradient Descent (SGD) to find the initial parameters. A non-zero value set specifies the tolerance SGD uses to determine convergence. The default value is `0` specifying that SGD is not used.

`trainThreads`

The number of threads to use in training the model. This should be set to the number of cores on the machine. Note that L-BFGS multi-threading attempts to load dataset into memory. In case of out-of-memory issues, set `trainThreads` to `1` to turn off multi-threading. If `NULL` the number of threads to use is determined internally. The default value is `NULL`.

`denseOptimizer`

If `TRUE`, forces densification of the internal optimization vectors. If `FALSE`, enables the logistic regression optimizer use sparse or dense internal states as it finds appropriate. Setting `denseOptimizer` to `TRUE` requires the internal optimizer to use a dense internal state, which may help alleviate load on the garbage collector for some varieties of larger problems.

`normalize`

Specifies the type of automatic normalization used:

- `"auto"`: if normalization is needed, it is performed automatically. This is the default choice.
- `"no"`: no normalization is performed.
- `"yes"`: normalization is performed.
- `"warn"`: if normalization is needed, a warning message is displayed, but normalization is not performed.

Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If

normalization is performed, a `MaxMin` normalizer is used. It normalizes values in an interval $[a, b]$ where $-1 \leq a \leq 0$ and $0 \leq b \leq 1$ and $b - a = 1$. This normalizer preserves sparsity by mapping zero to zero.

`m1Transforms`

Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See [featurizeText](#), [categorical](#), and [categoricalHash](#), for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.

`m1TransformVars`

Specifies a character vector of variable names to be used in `m1Transforms` or `NULL` if none are to be used. The default value is `NULL`.

`rowSelection`

Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

An expression of the form `list(name = expression, ``...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

`transformObjects`

A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

The variable transformation function. See `rxTransform` for details.

transformVars

A character vector of input data set variables needed for the transformation function. See `rxTransform` for details.

transformPackages

A character vector specifying additional R packages (outside of those specified in `rxgetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxgetOption("transformPackages")` are preloaded.

transformEnvir

A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

blocksPerRead

Specifies the number of blocks to read for each chunk of data read from the data source.

reportProgress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

computeContext

Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently `local` and `RxInSqlServer` compute contexts are supported.

ensemble

Control parameters for ensembling.



Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

Logistic Regression is a classification method used to predict the value of a categorical dependent variable from its relationship to one or more independent variables assumed to have a logistic distribution. If the dependent variable has only two possible values (success/failure), then the logistic regression is binary. If the dependent variable has more than two possible values (blood type given diagnostic test results), then the logistic regression is multinomial.

The optimization technique used for `rxLogisticRegression` is the limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS). Both the L-BFGS and regular BFGS algorithms use quasi-Newtonian methods to estimate the computationally intensive Hessian matrix in the equation used by Newton's method to calculate steps. But the L-BFGS approximation uses only a limited amount of memory to compute the next step direction, so that it is especially suited for problems with a large number of variables. The `memorySize` parameter specifies the number of past positions and gradients to store for use in the computation of the next step.

This learner can use elastic net regularization: a linear combination of L1 (lasso) and L2 (ridge) regularizations. Regularization is a method that can render an ill-posed problem more tractable by imposing constraints that provide information to supplement the data

and that prevents overfitting by penalizing models with extreme coefficient values. This can improve the generalization of the model learned by selecting the optimal complexity in the bias-variance tradeoff. Regularization works by adding the penalty that is associated with coefficient values to the error of the hypothesis. An accurate model with extreme coefficient values would be penalized more, but a less accurate model with more conservative values would be penalized less. L1 and L2 regularization have different effects and uses that are complementary in certain respects.

`L1Weight`: can be applied to sparse models, when working with high-dimensional data. It pulls small weights associated features that are relatively unimportant towards 0.

`L2Weight`: is preferable for data that is not sparse. It pulls large weights towards zero.

Adding the ridge penalty to the regularization overcomes some of lasso's limitations. It can improve its predictive accuracy, for example, when the number of predictors is greater than the sample size. If `x = L1Weight` and `y = L2Weight`, `ax + by = c` defines the linear span of the regularization terms. The default values of x and y are both 1. An aggressive regularization can harm predictive capacity by excluding important variables out of the model. So choosing the optimal values for the regularization parameters is important for the performance of the logistic regression model.

Value

`rxLogisticRegression`: A `rxLogisticRegression` object with the trained model.

`LogisticReg`: A learner specification object of class `mam1` for the Logistic Reg trainer.

Notes

This algorithm will attempt to load the entire dataset into memory when `trainThreads > 1` (multi-threading).

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) ↗

References

[Wikipedia: L-BFGS](#) ↗

[regression](#) ↗

[Training of L1-Regularized Log-Linear Models](#) ↗

[and L2 Regularization for Machine Learning](#)

See also

[rxFastTrees](#), [rxFastForest](#), [rxFastLinear](#), [rxNeuralNet](#), [rxOneClassSvm](#), [featurizeText](#), [categorical](#), [categoricalHash](#), [rxPredict.mlModel](#).

Examples

```
# Estimate a logistic regression model
logitModel <- rxLogisticRegression(isCase ~ age + parity + education +
  spontaneous + induced,
  transforms = list(isCase = case == 1),
  data = infert)
# Print a summary of the model
summary(logitModel)

# Score to a data frame
scoreDF <- rxPredict(logitModel, data = infert,
  extraVarsToWrite = "isCase")

# Compute and plot the Radio Operator Curve and AUC
roc1 <- rxRoc(actualVarName = "isCase", predVarNames = "Probability", data
= scoreDF)
plot(roc1)
rxAuc(roc1)

#####
#####
# Multi-class logistic regression
testObs <- rnorm(nrow(iris)) > 0
testIris <- iris[testObs,]
trainIris <- iris[!testObs,]
multiLogit <- rxLogisticRegression(
  formula = Species~Sepal.Length + Sepal.Width + Petal.Length +
Petal.Width,
  type = "multiClass", data = trainIris)

# Score the model
scoreMultiDF <- rxPredict(multiLogit, data = testIris,
  extraVarsToWrite = "Species")
# Print the first rows of the data frame with scores
```

```
head(scoreMultiDF)
# Look at confusion matrix
table(scoreMultiDF$Species, scoreMultiDF$PredictedLabel)

# Look at the observations with incorrect predictions
badPrediction = scoreMultiDF$Species != scoreMultiDF$PredictedLabel
scoreMultiDF[badPrediction,]
```


rxNeuralNet: Neural Net

Article • 03/01/2023

Neural networks for regression modeling and for Binary and multi-class classification.

Usage

```
rxNeuralNet(formula = NULL, data, type = c("binary", "multiClass",
      "regression"), numHiddenNodes = 100, numIterations = 100,
      optimizer = sgd(), netDefinition = NULL, initWtsDiameter = 0.1,
      maxNorm = 0, acceleration = c("sse", "gpu"), miniBatchSize = 1,
      normalize = "auto", mlTransforms = NULL, mlTransformVars = NULL,
      rowSelection = NULL, transforms = NULL, transformObjects = NULL,
      transformFunc = NULL, transformVars = NULL, transformPackages = NULL,
      transformEnvir = NULL, blocksPerRead = rxGetOption("blocksPerRead"),
      reportProgress = rxGetOption("reportProgress"), verbose = 1,
      computeContext = rxGetOption("computeContext"),
      ensemble = ensembleControl(), ...)
```

Arguments

formula

The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.

data

A data source object or a character string specifying a `.xdf` file or a data frame object.

type

A character string denoting Fast Tree type:

- `"binary"` for the default binary classification neural network.
- `"multiClass"` for multi-class classification neural network.
- `"regression"` for a regression neural network.

numHiddenNodes

The default number of hidden nodes in the neural net. The default value is 100.

numIterations

The number of iterations on the full training set. The default value is 100.

optimizer

A list specifying either the `sgd` or `adaptive` optimization algorithm. This list can be created using `sgd` or `adaDeltaSgd`. The default value is `sgd`.

netDefinition

The Net# definition of the structure of the neural network. For more information about the Net# language, see [Reference Guide](#)

initWtsDiameter

Sets the initial weights diameter that specifies the range from which values are drawn for the initial learning weights. The weights are initialized randomly from within this range. The default value is 0.1.

maxNorm

Specifies an upper bound to constrain the norm of the incoming weight vector at each hidden unit. This can be very important in maxout neural networks as well as in cases where training produces unbounded weights.

acceleration

Specifies the type of hardware acceleration to use. Possible values are "sse" and "gpu". For GPU acceleration, it is recommended to use a `miniBatchSize` greater than one. If you want to use the GPU acceleration, there are additional manual setup steps are required:

- Download and install NVidia CUDA Toolkit 6.5 ([CUDA Toolkit](#) ↗).
- Download and install NVidia cuDNN v2 Library ([cudnn Library](#) ↗).
- Find the `libs` directory of the MicrosoftRML package by calling

```
system.file("mxLibs/x64", package = "MicrosoftML").
```

- Copy cublas64_65.dll, cudart64_65.dll and cusparse64_65.dll from the CUDA Toolkit 6.5 into the libs directory of the MicrosoftML package.
- Copy cudnn64_65.dll from the cuDNN v2 Library into the libs directory of the MicrosoftML package.

miniBatchSize

Sets the mini-batch size. Recommended values are between 1 and 256. This parameter is only used when the acceleration is GPU. Setting this parameter to a higher value improves the speed of training, but it might negatively affect the accuracy. The default value is 1.

normalize

Specifies the type of automatic normalization used:

- "auto": if normalization is needed, it is performed automatically. This is the default choice.
- "no": no normalization is performed.
- "yes": normalization is performed.
- "warn": if normalization is needed, a warning message is displayed, but normalization is not performed.

Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MinMax` normalizer is used. It normalizes values in an interval $[a, b]$ where $-1 \leq a \leq 0$ and $0 \leq b \leq 1$ and $b - a = 1$. This normalizer preserves sparsity by mapping zero to zero.

m1Transforms

Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See [featurizeText](#), [categorical](#), and [categoricalHash](#), for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.

m1TransformVars

Specifies a character vector of variable names to be used in `m1Transforms` or `NULL` if none are to be used. The default value is `NULL`.

rowSelection

Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

transforms

An expression of the form `list(name = expression, ``...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

transformObjects

A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

The variable transformation function. See `rxTransform` for details.

transformVars

A character vector of input data set variables needed for the transformation function. See `rxTransform` for details.

transformPackages

A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in `RevoScaleR` functions via their `transforms` and `transformFunc` arguments or those defined implicitly

via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.

transformEnvir

A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

blocksPerRead

Specifies the number of blocks to read for each chunk of data read from the data source.

reportProgress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

computeContext

Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently `local` and `RxInSqlServer` compute contexts are supported.

ensemble

Control parameters for ensembling.



Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

A neural network is a class of prediction models inspired by the human brain. A neural network can be represented as a weighted directed graph. Each node in the graph is called a neuron. The neurons in the graph are arranged in layers, where neurons in one layer are connected by a weighted edge (weights can be 0 or positive numbers) to neurons in the next layer. The first layer is called the input layer, and each neuron in the input layer corresponds to one of the features. The last layer of the function is called the output layer. So in the case of binary neural networks it contains two output neurons, one for each class, whose values are the probabilities of belonging to each class. The remaining layers are called hidden layers. The values of the neurons in the hidden layers and in the output layer are set by calculating the weighted sum of the values of the neurons in the previous layer and applying an activation function to that weighted sum. A neural network model is defined by the structure of its graph (namely, the number of hidden layers and the number of neurons in each hidden layer), the choice of activation function, and the weights on the graph edges. The neural network algorithm tries to learn the optimal weights on the edges based on the training data.

Although neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are also easily adapted to regression problems. Any class of statistical models can be considered a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Neural network regression is especially suited to problems where a more traditional regression model cannot fit a solution.

Value

`rxNeuralNet`: an `rxNeuralNet` object with the trained model.

`NeuralNet`: a learner specification object of class `mam1` for the Neural Net trainer.

Notes

This algorithm is single-threaded and will not attempt to load the entire dataset into memory.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

References

[Wikipedia: Artificial neural network](#) 

See also

[rxFastTrees](#), [rxFastForest](#), [rxFastLinear](#), [rxLogisticRegression](#), [rxOneClassSvm](#), [featurizeText](#), [categorical](#), [categoricalHash](#), [rxPredict.mlModel](#).

Examples

```
# Estimate a binary neural net
rxNeuralNet1 <- rxNeuralNet(isCase ~ age + parity + education + spontaneous
+ induced,
                           transforms = list(isCase = case == 1),
                           data = infert)

# Score to a data frame
scoreDF <- rxPredict(rxNeuralNet1, data = infert,
                    extraVarsToWrite = "isCase",
                    outData = NULL) # return a data frame

# Compute and plot the Radio Operator Curve and AUC
roc1 <- rxRoc(actualVarName = "isCase", predVarNames = "Probability", data
= scoreDF)
plot(roc1)
rxAuc(roc1)

#####
# Regression neural net

# Create an xdf file with the attitude data
myXdf <- tempfile(pattern = "tempAttitude", fileext = ".xdf")
rxDataStep(attitude, myXdf, rowsPerRead = 50, overwrite = TRUE)
myXdfDS <- RxXdfData(file = myXdf)

attitudeForm <- rating ~ complaints + privileges + learning +
  raises + critical + advance

# Estimate a regression neural net
```

```

res2 <- rxNeuralNet(formula = attitudeForm, data = myXdfDS,
  type = "regression")

# Score to data frame
scoreOut2 <- rxPredict(res2, data = myXdfDS,
  extraVarsToWrite = "rating")

# Plot the rating versus the score with a regression line
rxLinePlot(rating~Score, type = c("p","r"), data = scoreOut2)

# Clean up
file.remove(myXdf)

#####
#
# Multi-class neural net
multiNN <- rxNeuralNet(
  formula = Species~Sepal.Length + Sepal.Width + Petal.Length +
Petal.Width,
  type = "multiClass", data = iris)
scoreMultiDF <- rxPredict(multiNN, data = iris,
  extraVarsToWrite = "Species", outData = NULL)
# Print the first rows of the data frame with scores
head(scoreMultiDF)
# Compute % of incorrect predictions
badPrediction = scoreMultiDF$Species != scoreMultiDF$PredictedLabel
sum(badPrediction)*100/nrow(scoreMultiDF)
# Look at the observations with incorrect predictions
scoreMultiDF[badPrediction,]

```


rxOneClassSvm: OneClass SVM

Article • 02/28/2023

Machine Learning One Class Support Vector Machines

Usage

```
rxOneClassSvm(formula = NULL, data, cacheSize = 100, kernel = rbfKernel(),
  epsilon = 0.001, nu = 0.1, shrink = TRUE, normalize = "auto",
  mlTransforms = NULL, mlTransformVars = NULL, rowSelection = NULL,
  transforms = NULL, transformObjects = NULL, transformFunc = NULL,
  transformVars = NULL, transformPackages = NULL, transformEnvir = NULL,
  blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 1,
  computeContext = rxGetOption("computeContext"),
  ensemble = ensembleControl(), ...)
```

Arguments

formula

The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.

data

A data source object or a character string specifying a `.xdf` file or a data frame object.

cacheSize

The maximal size in MB of the cache that stores the training data. Increase this for large training sets. The default value is 100 MB.

kernel

A character string representing the kernel used for computing inner products. For more information, see [makeKernel](#). The following choices are available:

- `rbfKernel()`: Radial basis function kernel. Its parameter represents `gamma` in the term $\exp(-\text{gamma}|x-y|^2)$. If not specified, it defaults to `1` divided by the number of features used. For example, `rbfKernel(gamma = .1)`. This is the default value.
- `linearKernel()`: Linear kernel.
- `polynomialKernel()`: Polynomial kernel with parameter names `a`, `bias`, and `deg` in the term $(a \cdot \langle x, y \rangle + \text{bias})^{\text{deg}}$. The `bias`, defaults to `0`. The degree, `deg`, defaults to `3`. If `a` is not specified, it is set to `1` divided by the number of features. For example, `makeKernelPolynomial(bias = 0, deg = 3)`.
- `sigmoidKernel()`: Sigmoid kernel with parameter names `gamma` and `coef0` in the term $\tanh(\text{gamma} \cdot \langle x, y \rangle + \text{coef0})$. `gamma`, defaults to `1` divided by the number of features. The parameter `coef0` defaults to `0`. For example, `sigmoidKernel(gamma = .1, coef0 = 0)`.

epsilon

The threshold for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model. The value must be greater than or equal to `.Machine$double.eps`. The default value is 0.001.

nu

The trade-off between the fraction of outliers and the number of support vectors (represented by the Greek letter nu). Must be between 0 and 1, typically between 0.1 and 0.5. The default value is 0.1.

shrink

Uses the shrinking heuristic if `TRUE`. In this case, some samples will be "shrunk" during the training procedure, which may speed up training. The default value is `TRUE`.

normalize

Specifies the type of automatic normalization used:

- `"auto"`: if normalization is needed, it is performed automatically. This is the default choice.
- `"no"`: no normalization is performed.
- `"yes"`: normalization is performed.

- "warn": if normalization is needed, a warning message is displayed, but normalization is not performed.

Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MinMax` normalizer is used. It normalizes values in an interval $[a, b]$ where $-1 \leq a \leq 0$ and $0 \leq b \leq 1$ and $b - a = 1$. This normalizer preserves sparsity by mapping zero to zero.

`m1Transforms`

Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See [featurizeText](#), [categorical](#), and [categoricalHash](#), for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.

`m1TransformVars`

Specifies a character vector of variable names to be used in `m1Transforms` or `NULL` if none are to be used. The default value is `NULL`.

`rowSelection`

Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20) & (age < 65) & (log(income) > 10)` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the expression function.

`transforms`

An expression of the form `list(name = expression, ``...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the expression function.

transformObjects

A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

transformFunc

The variable transformation function. See `rxTransform` for details.

transformVars

A character vector of input data set variables needed for the transformation function. See `rxTransform` for details.

transformPackages

A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For example, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.

transformEnvir

A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

blocksPerRead

Specifies the number of blocks to read for each chunk of data read from the data source.

reportProgress

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.

- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

computeContext

Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently `local` and `RxInSqlServer` compute contexts are supported.

ensemble

Control parameters for ensembling.

...

Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

detection is to identify outliers that do not belong to some target class. This type of SVM is one-class because the training set contains only examples from the target class. It infers what properties are normal for the objects in the target class and from these properties predicts which examples are unlike the normal examples. This is useful for anomaly detection because the scarcity of training examples is the defining character of anomalies: typically there are very few examples of network intrusion, fraud, or other types of anomalous behavior.

Value

`rxOneClassSvm`: A `rxOneClassSvm` object with the trained model.

`OneClassSvm`: A learner specification object of class `mam1` for the OneClass Svm trainer.

Notes

This algorithm is single-threaded and will always attempt to load the entire dataset into memory.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

References

[Anomaly detection](#) 

[Azure Machine Learning Studio \(classic\): One-Class Support Vector Machine](#)

[Support of a High-Dimensional Distribution](#) 

[Support Vector Algorithms](#) 

[for Support Vector Machines](#) 

See also

[rbfKernel](#), [linearKernel](#), [polynomialKernel](#), [sigmoidKernel](#) [rxFastTrees](#), [rxFastForest](#), [rxFastLinear](#), [rxLogisticRegression](#), [rxNeuralNet](#), [featurizeText](#), [categorical](#), [categoricalHash](#), [rxPredict.mlModel](#).

Examples

```
# Estimate a One-Class SVM model
trainRows <- c(1:30, 51:80, 101:130)
testRows = !(1:150 %in% trainRows)
trainIris <- iris[trainRows,]
testIris <- iris[testRows,]

svmModel <- rxOneClassSvm(
  formula = ~Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
  data = trainIris)

# Add additional non-iris data to the test data set
testIris$isIris <- 1
```

```
notIris <- data.frame(  
  Sepal.Length = c(2.5, 2.6),  
  Sepal.Width = c(.75, .9),  
  Petal.Length = c(2.5, 2.5),  
  Petal.Width = c(.8, .7),  
  Species = c("not iris", "not iris"),  
  isIris = 0)  
testIris <- rbind(testIris, notIris)  
  
scoreDF <- rxPredict(svmModel,  
  data = testIris, extraVarsToWrite = "isIris")  
  
# Look at the last few observations  
tail(scoreDF)  
# Look at average scores conditioned by 'isIris'  
rxCube(Score ~ F(isIris), data = scoreDF)
```

rxPredict.mlModel: Score using a Microsoft R Machine Learning model

Article • 02/28/2023

Reports per-instance scoring results in a data frame or RevoScaleR data source using a trained Microsoft R Machine Learning model with a RevoScaleR data source.

Usage

```
## S3 method for class `mlModel':
rxPredict (modelObject, data, outData = NULL,
          writeModelVars = FALSE, extraVarsToWrite = NULL, suffix = NULL,
          overwrite = FALSE, dataThreads = NULL,
          blocksPerRead = rxGetOption("blocksPerRead"),
          reportProgress = rxGetOption("reportProgress"), verbose = 1,
          computeContext = rxGetOption("computeContext"), ...)
```

Arguments

modelObject

A model information object returned from a MicrosoftML model. For example, an object returned from [rxFastTrees](#) or [rxLogisticRegression](#).

data

A **RevoScaleR** data source object, a data frame, or the path to a **.xdf** file.

outData

Output text or xdf file name or an **RxDataSource** with write capabilities in which to store predictions. If **NULL**, a data frame is returned. The default value is **NULL**.

writeModelVars

If `TRUE`, variables in the model are written to the output data set in addition to the scoring variables. If variables from the input data set are transformed in the model, the transformed variables are also included. The default value is `FALSE`.

`extraVarsToWrite`

`NULL` or character vector of additional variables names from the input data to include in the `outData`. If `writeModelVars` is `TRUE`, model variables are included as well. The default value is `NULL`.

`suffix`

A character string specifying suffix to append to the created scoring variable(s) or `NULL` if there is no suffix. The default value is `NULL`.

`overwrite`

If `TRUE`, an existing `outData` is overwritten; if `FALSE` an existing `outData` is not overwritten. The default value is `FALSE`.

`dataThreads`

An integer specifying the desired degree of parallelism in the data pipeline. If `NULL`, the number of threads used is determined internally. The default value is `NULL`.

`blocksPerRead`

Specifies the number of blocks to read for each chunk of data read from the data source.

`reportProgress`

An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

The default value is `1`.

verbose

An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information. The default value is `1`.

computeContext

Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently `local` and `RxInSqlServer` compute contexts are supported.

...

Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

The following items are reported in the output by default: scoring on three variables for the binary classifiers: `PredictedLabel`, `Score`, and `Probability`; the `Score` for `oneClassSvm` and regression classifiers; `PredictedLabel` for Multi-class classifiers, plus a variable for each category prepended by the `Score`.

Value

A data frame or an `RxDataSource` object representing the created output data. By default, output from scoring binary classifiers include three variables: `PredictedLabel`, `Score`, and `Probability`; `rxOneClassSvm` and regression include one variable: `Score`; and multi-class classifiers include `PredictedLabel` plus a variable for each category prepended by `Score`. If a `suffix` is provided, it is added to the end of these output variable names.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

See also

[rxFastTrees](#), [rxFastForest](#), [rxLogisticRegression](#), [rxNeuralNet](#), [rxOneClassSvm](#).

Examples

```
# Estimate a logistic regression model
infert1 <- infert
infert1$isCase <- (infert1$case == 1)
myModelInfo <- rxLogisticRegression(formula = isCase ~ age + parity +
education + spontaneous + induced,
data = infert1)

# Create an xdf file with per-instance results using rxPredict
xdfOut <- tempfile(pattern = "scoreOut", fileext = ".xdf")
scoreDS <- rxPredict(myModelInfo, data = infert1,
outData = xdfOut, overwrite = TRUE,
extraVarsToWrite = c("isCase", "Probability"))

# Summarize results with an ROC curve
rxRocCurve(actualVarName = "isCase", predVarNames = "Probability", data =
scoreDS)

# Use the built-in data set 'airquality' to create test and train data
DF <- airquality[!is.na(airquality$Ozone), ]
DF$Ozone <- as.numeric(DF$Ozone)
set.seed(12)
randomSplit <- rnorm(nrow(DF))
trainAir <- DF[randomSplit >= 0,]
testAir <- DF[randomSplit < 0,]
airFormula <- Ozone ~ Solar.R + Wind + Temp

# Regression Fast Tree for train data
fastTreeReg <- rxFastTrees(airFormula, type = "regression",
data = trainAir)

# Put score and model variables in data frame, including the model
variables
# Add the suffix "Pred" to the new variable
fastTreeScoreDF <- rxPredict(fastTreeReg, data = testAir,
writeModelVars = TRUE, suffix = "Pred")

rxGetVarInfo(fastTreeScoreDF)

# Clean-up
file.remove(xdfOut)
```

selectColumns: Selects a set of columns, dropping all others

Article • 02/28/2023

Selects a set of columns to retrain, dropping all others.

Usage

```
selectColumns(vars, ...)
```

Arguments

vars

Specifies character vector or list of the names of the variables to keep.

...

Additional arguments sent to compute engine.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation [Microsoft Technical Support](#) 

selectFeatures: Machine Learning Feature Selection Transform

Article • 02/28/2023

The feature selection transform selects features from the specified variables using the specified mode.

Usage

```
selectFeatures(vars, mode, ...)
```

Arguments

vars

A formula or a vector/list of strings specifying the name of variables upon which the feature selection is performed, if the mode is `minCount()`. For example, `~ var1 + var2 + var3`. If mode is `mutualInformation()`, a formula or a named list of strings describing the dependent variable and the independent variables. For example, `label ~ `var1 + var2 + var3``.

mode

Specifies the mode of feature selection. This can be either [minCount](#) or [mutualInformation](#).

...

Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

The feature selection transform selects features from the specified variables using one of the two modes: count or mutual information. For more information, see [minCount](#) and

[mutualInformation](#).

Value

A `mam1` object defining the transform.

See also

[minCount mutualInformation](#)

Examples

```
trainReviews <- data.frame(review = c(
  "This is great",
  "I hate it",
  "Love it",
  "Do not like it",
  "Really like it",
  "I hate it",
  "I like it a lot",
  "I kind of hate it",
  "I do like it",
  "I really hate it",
  "It is very good",
  "I hate it a bunch",
  "I love it a bunch",
  "I hate it",
  "I like it very much",
  "I hate it very much.",
  "I really do love it",
  "I really do hate it",
  "Love it!",
  "Hate it!",
  "I love it",
  "I hate it",
  "I love it",
  "I hate it",
  "I love it"),
  like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
  FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
)

testReviews <- data.frame(review = c(
  "This is great",
```

```

    "I hate it",
    "Love it",
    "Really like it",
    "I hate it",
    "I like it a lot",
    "I love it",
    "I do like it",
    "I really hate it",
    "I love it"), stringsAsFactors = FALSE)

# Use a categorical hash transform which generated 128 features.
outModel1 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
l1Weight = 0,
  mlTransforms = list(categoricalHash(vars = c(reviewCatHash = "review"),
hashBits = 7)))
summary(outModel1)

# Apply a categorical hash transform and a count feature selection
transform
# which selects only those hash slots that has value.
outModel2 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
l1Weight = 0,
  mlTransforms = list(
    categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
    selectFeatures("reviewCatHash", mode = minCount()))))
summary(outModel2)

# Apply a categorical hash transform and a mutual information feature
selection transform
# which selects only 10 features with largest mutual information with the
label.
outModel3 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
l1Weight = 0,
  mlTransforms = list(
    categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
    selectFeatures(like ~ reviewCatHash, mode =
mutualInformation(numFeaturesToKeep = 10))))
summary(outModel3)

```

olapR (R package in SQL Server Machine Learning Services)

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x) and later versions

olapR is an R package from Microsoft used for MDX queries against a SQL Server Analysis Services OLAP cube. Functions do not support all MDX operations, but you can build queries that slice, dice, drilldown, rollup, and pivot on dimensions. The package is included in [SQL Server Machine Learning Services](#) and [SQL Server 2016 R Services](#).

You can use this package on connections to an Analysis Services OLAP cube on all supported versions of SQL Server. Connections to a tabular model are not supported at this time.

Load package

The **olapR** package is not preloaded into an R session. Run the following command to load the package.

```
R  
  
library(olapR)
```

Package version

Current version is 1.0.0 in all Windows-only products and downloads providing the package.

Availability and location

This package is provided in the following products, as well as on several virtual machine images on Azure. Package location varies accordingly.

Product	Location
SQL Server Machine Learning Services (with R integration)	C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library

Product	Location
SQL Server 2016 R Services	C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\R_SERVICES\library
Microsoft Machine Learning Server (R Server)	C:\Program Files\Microsoft\R_SERVER\library
Microsoft R Client	C:\Program Files\Microsoft\R Client\R_SERVER\library
Data Science Virtual Machine (on Azure)	C:\Program Files\Microsoft\R Client\R_SERVER\library
SQL Server Virtual Machine (on Azure) ¹	C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\R_SERVICES\library

¹ R integration is optional in SQL Server. The `olapR` package will be installed when you add the Machine Learning or R feature during VM configuration.

How to use `olapR`

The `olapR` library provides a simple R style API for generating and validating MDX queries against an Analysis Services cube. `olapR` does not provide APIs for all MDX scenarios, but it does cover the most use cases including slice, dice, drilldown, rollup, and pivot scenarios in N dimensions. You can also input a direct MDX query to Analysis Services for queries that cannot be constructed using the `olapR` APIs.

Workflow for using `olapR`

1. Load the library.
2. Create a connection string pointing to a MOLAP cube on Analysis Services.
3. Verify you have read access on the cube
4. Use the connection string on a connection.
5. Verify the connection using the `explore` function.
6. Set up a query by submitting an MDX query string or by building a query structure.
7. Execute the query and verify the result.

To execute an MDX query on an OLAP Cube, you need to first create a connection string (`olapConn`) and validate using the function `olapConnection(connectionString)`. The connection string must have a Data Source (such as localhost) and a Provider (MSOLAP).

After the connection is established, you can either pass in a fully defined MDX query, or you can construct the query using the `Query()` object, setting the query details using `cube()`, `axis()`, `columns()`, `slicers()`, and so forth.

Finally, pass the `olapConn` and query into either `executeMD` or `execute2D` to get a multidimensional array or a data frame back.

📘 Important

`olapR` requires the Analysis Services OLE DB provider. If you do not have SQL Server Analysis Services installed on your computer, download the provider from Microsoft: [Data providers used for Analysis Services connections](#)

The exact version you should install for SQL Server 2016 is [here](#) ↗.

Function list

Function	Description
OlapConnection	Create the connection string to access the Analysis Services Database.
Query	Construct a Query object to use on the Analysis Services Database. Use cube, axis, columns, rows, pages, chapters, slicers to add details to the query.
executeMD	Takes a Query object or an MDX string, and returns the result as a multi-dimensional array.
execute2D	Takes a Query object or an MDX string, and returns the result as a 2D data frame.
explore	Allows for exploration of cube metadata.

MDX concepts

MDX is the query language for multidimensional OLAP (MOLAP) cubes containing processed and aggregated data stored in structures optimized for data analysis and exploration. Cubes are used in business and scientific applications to draw insights about relationships in historical data. Internally, cubes consist of mostly quantifiable numeric data, which is sliced along dimensions like date and time, geography, or other entities. A typical query might roll up sales for a given region and time period, sliced by product category, promotion, sales channel, and so forth.

Cube data can be accessed using various operations:

- Slicing - Taking a subset of the cube by picking a value for one dimension, resulting in a cube that is one dimension smaller.

- Dicing - Creating a subcube by specifying a range of values on multiple dimensions.
- Drill-Down/Up - Navigate from more general to more detailed data ranges, or vice versa.
- Roll-up - Summarize the data on a dimension.
- Pivot - Rotate the cube.

MDX queries are similar to SQL queries but, because of the flexibility of OLAP databases, can contain up to 128 query axes. The first four axes are named for convenience: Columns, Rows, Pages, and Chapters. It's also common to just use two (Rows and Columns), as shown in the following example:

SQL

```
SELECT {[Measures].[Internet Sales Count], [Measures].[Internet Sales-Sales Amount]} ON COLUMNS,
{[Product].[Product Line].[Product Line].MEMBERS} ON ROWS
FROM [Analysis Services Tutorial]
WHERE [Sales Territory].[Sales Territory Country].[Australia]
```

Using an AdventureWorks OLAP cube from the [multidimensional cube tutorial](#), this MDX query selects the internet sales count and sales amount and places them on the Column axis. On the Row axis it places all possible values of the "Product Line" dimension. Then, using the WHERE clause (which is the slicer axis in MDX queries), it filters the query so that only the sales from Australia matter. Without the slicer axis, we would roll up and summarize the sales from all countries/regions.

olapR examples

R

```
# load the library
library(olapR)

# Connect to a local SSAS default instance and the Analysis Services Tutorial database.
# For named instances, use server-name\\instancename, escaping the instance name delimiter.
# For databases containing multiple cubes, use the cube= parameter to specify which one to use.
cnstr <- "Data Source=localhost; Provider=MSOLAP; initial catalog=Analysis Services Tutorial"
olapCnn <- OlapConnection(cnstr)
```

```
# Approach 1 - build the mdx query in R
qry <- Query()

cube(qry) <- "[Analysis Services Tutorial]"
columns(qry) <- c("[Measures].[Internet Sales Count]", "[Measures].[Internet
Sales-Sales Amount]")
rows(qry) <- c("[Product].[Product Line].[Product Line].MEMBERS")
slicers(qry) <- c("[Sales Territory].[Sales Territory Country].[Australia]")

result1 <- executeMD(olapCnn, qry)

# Approach 2 - Submit a fully formed MDX query
mdx <- "SELECT {[Measures].[Internet Sales Count], [Measures].[Internet
Sales-Sales Amount]} ON AXIS(0), {[Product].[Product Line].[Product
Line].MEMBERS} ON AXIS(1) FROM [Analysis Services Tutorial] WHERE [Sales
Territory].[Sales Territory Country].[Australia]"

result2 <- execute2D(olapCnn, mdx)
```

See also

[How to create MDX queries using olapR](#)

execute2D: olapR execute2D Methods

Article • 02/28/2023

Takes a Query object or an MDX string, and returns the result as a data frame.

Usage

```
execute2D(olapCnn, query)  
execute2D(olapCnn, mdx)
```

Arguments

olapCnn

Object of class "OlapConnection" returned by `olapConnection()`

query

Object of class "Query" returned by `Query()`

mdx

String specifying a valid MDX query

Details

If a query is provided: `execute2D` validates a query object (optional), generates an mdx query string from the query object, executes the mdx query across, and returns the result as a data frame.

If an MDX string is provided: `execute2D` executes the mdx query, and returns the result as a data frame.

Value

A data frame if the MDX command returned a result-set. `TRUE` and a warning if the query returned no data. An error if the query is invalid

Notes

Multi-dimensional query results are flattened to 2D using a standard flattening algorithm.

References

Creating a Demo OLAP Cube (the same as the one used in the examples):

- [Multidimensional Modeling \(Adventure Works Tutorial\)](#)

See also

[Query](#), [OlapConnection](#), [executeMD](#), [explore](#), [data.frame](#)

Examples

```
cnnstr <- "Data Source=localhost; Provider=MSOLAP;"
olapCnn <- OlapConnection(cnnstr)

qry <- Query()

cube(qry) <- "[Analysis Services Tutorial]"
columns(qry) <- c("[Measures].[Internet Sales Count]", "[Measures].[Internet Sales-Sales Amount]")
rows(qry) <- c("[Product].[Product Line].[Product Line].MEMBERS")
pages(qry) <- c("[Sales Territory].[Sales Territory Region].[Sales Territory Region].MEMBERS")

result1 <- execute2D(olapCnn, qry)

mdx <- "SELECT {[Measures].[Internet Sales Count], [Measures].[Internet Sales-Sales Amount]} ON AXIS(0), {[Product].[Product Line].[Product Line].MEMBERS} ON AXIS(1), {[Sales Territory].[Sales Territory Region].[Sales Territory Region].MEMBERS} ON AXIS(2) FROM [Analysis Services Tutorial]"

result2 <- execute2D(olapCnn, mdx)
```

executeMD: olapR executeMD Methods

Article • 02/28/2023

Takes a Query object or an MDX string, and returns the result as a multi-dimensional array.

Usage

```
executeMD(olapCnn, query)
executeMD(olapCnn, mdx)
```

Arguments

olapCnn

Object of class "OlapConnection" returned by `olapConnection()`

query

Object of class "Query" returned by `Query()`

mdx

String specifying a valid MDX query

Details

If a Query is provided: `executeMD` validates a Query object (optional), generates an mdx query string from the Query object, executes the mdx query across an XMLA connection, and returns the result as a multi-dimensional array.

If an MDX string is provided: `executeMD` executes the mdx query across an XMLA connection, and returns the result as a multi-dimensional array.

Value

Returns a multi-dimensional array. Returns an error if the Query is invalid.

Notes

References

Creating a Demo OLAP Cube (the same as the one used in the examples):

[Multidimensional Modeling \(Adventure Works Tutorial\)](#)

See also

[Query](#), [OlapConnection](#), [execute2D](#), [explore](#), [array](#)

Examples

```
cnnstr <- "Data Source=localhost; Provider=MSOLAP;"
olapCnn <- OlapConnection(cnnstr)

qry <- Query()

cube(qry) <- "[Analysis Services Tutorial]"
columns(qry) <- c("[Measures].[Internet Sales Count]", "[Measures].[Internet Sales-Sales Amount]")
rows(qry) <- c("[Product].[Product Line].[Product Line].MEMBERS")
pages(qry) <- c("[Sales Territory].[Sales Territory Region].[Sales Territory Region].MEMBERS")

result1 <- executeMD(olapCnn, qry)

mdx <- "SELECT {[Measures].[Internet Sales Count], [Measures].[Internet Sales-Sales Amount]} ON AXIS(0), {[Product].[Product Line].[Product Line].MEMBERS} ON AXIS(1), {[Sales Territory].[Sales Territory Region].[Sales Territory Region].MEMBERS} ON AXIS(2) FROM [Analysis Services Tutorial]"

result2 <- executeMD(olapCnn, mdx)
```


explore: olapR explore Method

Article • 02/28/2023

Allows for exploration of cube metadata

Usage

```
explore(olapCnn, cube = NULL, dimension = NULL, hierarchy = NULL, level = NULL)
```

Arguments

olapCnn

Object of class "OlapConnection" returned by `olapConnection()`

cube

A string specifying a cube name

dimension

A string specifying a dimension name

hierarchy

A string specifying a hierarchy name

level

A string specifying a level name

Details

`explore`

Value

Prints cube metadata. Returns NULL. An error is thrown if arguments are invalid.

Notes

Arguments must be specified in order. For example: In order to explore hierarchies, a dimension and a cube must be specified.

References

See [execute2D](#) or [executeMD](#) for references.

See also

[query](#), [OlapConnection](#), [executeMD](#), [execute2D](#)

Examples

```
cnstr <- "Data Source=localhost; Provider=MSOLAP;"
ocs <- OlapConnection(cnstr)

#Exploring Cubes
explore(ocs)
#Analysis Services Tutorial
#Internet Sales
#Reseller Sales
#Sales Summary
#[1] TRUE

#Exploring Dimensions
explore(ocs, "Analysis Services Tutorial")
#Customer
#Date
#Due Date
#Employee
#Internet Sales Order Details
#Measures
#Product
#Promotion
#Reseller
#Reseller Geography
#Sales Reason
```

```
#Sales Territory
#Ship Date
#[1] TRUE

#Exploring Hierarchies
explore(ocs, "Analysis Services Tutorial", "Product")
#Category
#Class
#Color
#Days To Manufacture
#Dealer Price
#End Date
#List Price
#Model Name
#Product Categories
#Product Line
#Product Model Lines
#Product Name
#Reorder Point
#Safety Stock Level
#Size
#Size Range
#Standard Cost
#Start Date
#Status
#Style
#Subcategory
#Weight
#[1] TRUE

#Exploring Levels
explore(ocs, "Analysis Services Tutorial", "Product", "Product Categories")
#(All)
#Category
#Subcategory
#Product Name
#[1] TRUE

#Exploring Members
#NOTE: -> indicates that the following member is a child of the previous
member
explore(ocs, "Analysis Services Tutorial", "Product", "Product Categories",
"Category")
#Accessories
#Bikes
#Clothing
#Components
#Assembly Components
#-> Assembly Components
#--> Assembly Components
```

OlapConnection: olapR OlapConnection Creation

Article • 02/28/2023

`OlapConnection` constructs a "OlapConnection" object.

Usage

```
OlapConnection(connectionString="Data Source=localhost; Provider=MSOLAP;")  
  
is.OlapConnection(ocs)  
  
print.OlapConnection(ocs)
```

Arguments

`connectionString`

A valid connection string for connecting to Analysis Services

`ocs`

An object of class "OlapConnection"

Details

`OlapConnection` validates and holds an Analysis Services connection string. By default, Analysis Services returns the first cube of the first database. To connect to a specific database, use the Initial Catalog parameter.

Value

`OlapConnection` returns an object of type "OlapConnection". A warning is shown if the connection string is invalid.

References

For more information on Analysis Services connection strings, see [Connection string properties](#).

See also

[Query](#), [executeMD](#), [execute2D](#), [explore](#)

Examples

```
# Create the connection string. For a named instance, escape the instance
name: localhost\my-other-instance
cnstr <- "Data Source=localhost; Provider=MSOLAP; initial
catalog=AdventureWorksCube"
olapCnn <- OlapConnection(cnstr)
```

Query: olapR Query Construction

Article • 02/28/2023

`Query` constructs a "Query" object. Set functions are used to build and modify the Query axes and cube name.

Usage

```
Query(validate = FALSE)

cube(qry)
cube(qry) <- cubeName

columns(qry)
columns(qry) <- axis

rows(qry)
rows(qry) <- axis

pages(qry)
pages(qry) <- axis

chapters(qry)
chapters(qry) <- axis

axis(qry, n)
axis(qry, n) <- axis

slicers(qry)
slicers(qry) <- axis

compose(qry)

is.Query(qry)
```

Arguments

`validate`

A logical (TRUE, FALSE, NA) specifying whether the Query should be validated during execution

qry

An object of class "Query" returned by `Query`

cubeName

A string specifying the name of the cube to query

axis

A vector of strings specifying an axis. See example below.

n

An integer representing the axis number to be set. `axis(qry, 1) == columns(qry)`, `axis(qry, 2) == pages(qry)`, etc.

Details

`Query` is the constructor for the Query object. Set functions are used to specify what the Query should return. Queries are passed to the `Execute2D` and `ExecuteMD` functions.

`compose` takes the Query object and generates an MDX string equivalent to the one that the Execute functions would generate and use.

Value

`query` returns an object of type "Query". `cube` returns a string. `columns` returns a vector of strings. `rows` returns a vector of strings. `pages` returns a vector of strings. `sections` returns a vector of strings. `axis` returns a vector of strings. `slicers` returns a vector of strings. `compose` returns a string. `is.Query` returns a boolean.

Notes

- A Query object is not as powerful as pure MDX. If the Query API is not sufficient, try using an MDX Query string with one of the Execute functions.

References

See [execute2D](#) or [executeMD](#) for references.

See also

[execute2D](#), [executeMD](#), [OlapConnection](#), [explore](#)

Examples

```
qry <- Query(validate = TRUE)

cube(qry) <- "[Analysis Services Tutorial]"

columns(qry) <- c("[Measures].[Internet Sales Count]", "[Measures].
[Internet Sales-Sales Amount]")
rows(qry) <- c("[Product].[Product Line].[Product Line].MEMBERS")
axis(qry, 3) <- c("[Date].[Calendar Quarter].MEMBERS")

slicers(qry) <- c("[Sales Territory].[Sales Territories].[Sales Territory
Region].[Northwest]")

print(cube(qry)) #[Analysis Services Tutorial]
print(axis(qry, 2)) #c("[Product].[Product Line].[Product Line].MEMBERS")

print(compose(qry)) #SELECT {[Measures].[Internet Sales Count],
[Measures].[Internet Sales-Sales Amount]} ON AXIS(0), {[Product].[Product
Line].[Product Line].MEMBERS} ON AXIS(1), {[Date].[Calendar
Quarter].MEMBERS} ON AXIS(2) FROM [Analysis Services Tutorial] WHERE {[Sales
Territory].[Sales Territories].[Sales Territory Region].[Northwest]}
```


sqlrutils (R package in SQL Server Machine Learning Services)

Article • 02/28/2023

Applies to:  SQL Server 2016 (13.x) and later versions

sqlrutils is an R package from Microsoft that provides a mechanism for R users to put their R scripts into a T-SQL stored procedure, register that stored procedure with a database, and run the stored procedure from an R development environment. The package is included in [SQL Server Machine Learning Services](#) and [SQL Server 2016 R Services](#).

By converting your R code to run within a single stored procedure, you can make more effective use of SQL Server R Services, which requires that R script be embedded as a parameter to [sp_execute_external_script](#). The **sqlrutils** package helps you build this embedded R script and set related parameters appropriately.

The **sqlrutils** package performs these tasks:

- Saves the generated T-SQL script as a string inside an R data structure
- Optionally, generate a .sql file for the T-SQL script, which you can edit or run to create a stored procedure
- Registers the newly created stored procedure with the SQL Server instance from your R development environment

You can also execute the stored procedure from an R environment, by passing well-formed parameters and processing the results. Or, you can use the stored procedure from SQL Server to support common database integration scenarios such as ETL, model training, and high-volume scoring.

Note

If you intend to run the stored procedure from an R environment by calling the *executeStoredProcedure* function, you must use an ODBC 3.8 provider, such as ODBC Driver 13 for SQL Server.

Full reference documentation

The **sqlrutils** package is distributed in multiple Microsoft products, but usage is the same whether you get the package in SQL Server or another product. Should any

product-specific behaviors exist, discrepancies will be noted in the function help page.

Functions list

The following section provides an overview of the functions that you can call from the `sqlrutils` package to develop a stored procedure containing embedded R code. For details of the parameters for each method or function, see the R help for the package:

```
help(package="sqlrutils")
```

Function	Description
executeStoredProcedure	Execute a SQL stored procedure.
getInputParameters	Get a list of input parameters to the stored procedure.
InputData	Defines the source of data in SQL Server that will be used in the R data frame. You specify the name of the data.frame in which to store the input data, and a query to get the data, or a default value. Only simple SELECT queries are supported.
InputParameter	Defines a single input parameter that will be embedded in the T-SQL script. You must provide the name of the parameter and its R data type.
OutputData	Generates an intermediate data object that is needed if your R function returns a list that contains a data.frame. The <i>OutputData</i> object is used to store the name of a single data.frame obtained from the list.
OutputParameter	Generates an intermediate data object that is needed if your R function returns a list. The <i>OutputParameter</i> object stores the name and data type of a single member of the list, assuming that member is not a data frame.
registerStoredProcedure	Register the stored procedure with a database.
setInputDataQuery	Assign a query to an input data parameter of the stored procedure.
setInputParameterValue	Assign a value to an input parameter of the stored procedure.
StoredProcedure	A stored procedure object.

How to use sqlrutils

The `sqlrutils` package functions must run on a computer having SQL Server Machine Learning with R. If you are working on a client workstation, set a remote compute context to shift execution to SQL Server. The workflow for using this package includes the following steps:

- Define stored procedure parameters (inputs, outputs, or both)
- Generate and register the stored procedure
- Execute the stored procedure

In an R session, load `sqlrutils` from the command line by typing `library(sqlrutils)`.

ⓘ Note

You can load this package on computer that does not have SQL Server (for example, on an R Client instance) if you change the compute context to SQL Server and execute the code in that compute context.

Define stored procedure parameters and inputs

`StoredProcedure` is the main constructor used to build the stored procedure. This constructor generates a *SQL Server Stored Procedure* object, and optionally creates a text file containing a query that can be used to generate the stored procedure using a T-SQL command.

Optionally, the `StoredProcedure` function can also register the stored procedure with the specified instance and database.

- Use the `func` argument to specify a valid R function. All the variables that the function uses must be defined either inside the function or be provided as input parameters. These parameters can include a maximum of one data frame.
- The R function must return either a data frame, a named list, or a NULL. If the function returns a list, the list can contain a maximum of one data.frame.
- Use the argument `spName` to specify the name of the stored procedure you want to create.
- You can pass in optional input and output parameters, using the objects created by these helper functions: `setInputData`, `setInputParameter`, and `setOutputParameter`.
- Optionally, use `filePath` to provide the path and name of a .sql file to create. You can run this file on the SQL Server instance to generate the stored procedure using T-SQL.
- To define the server and database where the stored procedure will be saved, use the arguments `dbName` and `connectionString`.

- To get a list of the *InputData* and *InputParameter* objects that were used to create a specific *StoredProcedure* object, call `getInputParameters`.
- To register the stored procedure with the specified database, use `registerStoredProcedure`.

The stored procedure object typically does not have any data or values associated with it, unless a default value was specified. Data is not retrieved until the stored procedure is executed.

Specify inputs and execute

- Use `setInputDataQuery` to assign a query to an *InputParameter* object. For example, if you have created a stored procedure object in R, you can use `setInputDataQuery` to pass arguments to the *StoredProcedure* function in order to execute the stored procedure with the desired inputs.
- Use `setInputValue` to assign specific values to a parameter stored as an *InputParameter* object. You then pass the parameter object and its value assignment to the *StoredProcedure* function to execute the stored procedure with the set values.
- Use `executeStoredProcedure` to execute a stored procedure defined as an *StoredProcedure* object. Call this function only when executing a stored procedure from R code. Do not use it when running the stored procedure from SQL Server using T-SQL.

ⓘ Note

The `executeStoredProcedure` function requires an ODBC 3.8 provider, such as ODBC Driver 13 for SQL Server.

See also

[How to create a stored procedure using sqlrutils](#)

Convert R code to a stored procedure using sqlrutils

Article • 11/18/2022

This article describes the steps for using the `sqlrutils` package to convert your R code to run as a T-SQL stored procedure. For best possible results, your code might need to be modified somewhat to ensure that all inputs can be parameterized.

Step 1. Rewrite R Script

For the best results, you should rewrite your R code to encapsulate it as a single function.

All variables used by the function should be defined inside the function, or should be defined as input parameters. See the [sample code](#) in this article.

Also, because the input parameters for the R function will become the input parameters of the SQL stored procedure, you must ensure that your inputs and outputs conform to the following type requirements:

Inputs

Among the input parameters, there can be at most one data frame.

The objects inside the data frame, as well as all other input parameters of the function, must be of the following R data types:

- POSIXct
- numeric
- character
- integer
- logical
- raw

If an input type is not one of the above types, it needs to be serialized and passed into the function as `raw`. In this case, the function must also include code to deserialize the input.

Outputs

The function can output one of the following:

- A data frame containing the supported data types. All objects in the data frame must use one of the supported data types.
- A named list, containing at most one data frame. All members of the list should use one of the supported data types.
- A NULL, if your function does not return any result

Step 2. Generate Required Objects

After your R code has been cleaned up and can be called as a single function, you will use the functions in the `sqlrutils` package to prepare the inputs and outputs in a form that can be passed to the constructor that actually builds the stored procedure.

`sqlrutils` provides functions that define the input data schema and type, and define the output data schema and type. It also includes functions that can convert R objects to the required output type. You might make multiple function calls to create the required objects, depending on the data types your code uses.

Inputs

If your function takes inputs, for each input, call the following functions:

- `setInputData` if the input is a data frame
- `setInputParameter` for all other input types

When you make each function call, an R object is created that you will later pass as an argument to `StoredProcedure`, to create the complete stored procedure.

Outputs

`sqlrutils` provides multiple functions for converting R objects such as lists to the `data.frame` required by SQL Server. If your function outputs a data frame directly, without first wrapping it into a list, you can skip this step. You can also skip the conversion this step if your function returns NULL.

When converting a list or getting a particular item from a list, choose from these functions:

- `setOutputData` if the variable to get from the list is a data frame
- `setOutputParameter` for all other members of the list

When you make each function call, an R object is created that you will later pass as an argument to `StoredProcedure`, to create the complete stored procedure.

Step 3. Generate the Stored Procedure

When all input and output parameters are ready, make a call to the `StoredProcedure` constructor.

Usage

```
StoredProcedure (func, spName, ..., filePath = NULL ,dbName = NULL,  
connectionString = NULL, batchSeparator = "GO")
```

To illustrate, assume that you want to create a stored procedure named `sp_rsampl` with these parameters:

- Uses an existing function `foosql`. The function was based on existing code in R function `foo`, but you rewrote the function to conform to the requirements as described in [this section](#), and named the updated function as `foosql`.
- Uses the data frame `queryinput` as input
- Generates as output a data frame with the R variable name, `sqloutput`
- You want to create the T-SQL code as a file in the `C:\Temp` folder, so that you can run it using SQL Server Management Studio later

R

```
StoredProcedure (foosql, sp_rsampl, queryinput, sqloutput, filePath =  
"C:\\Temp")
```

ⓘ Note

Because you are writing the file to the file system, you can omit the arguments that define the database connection.

The output of the function is a T-SQL stored procedure that can be executed on an instance of SQL Server 2016 (requires R Services) or SQL Server 2017 (requires Machine Learning Services with R).

For additional examples, see the package help, by calling `help(StoredProcedure)` from an R environment.

Step 4. Register and Run the Stored Procedure

There are two ways that you can run the stored procedure:

- Using T-SQL, from any client that supports connections to the SQL Server 2016 or SQL Server 2017 instance
- From an R environment

Both methods require that the stored procedure be registered in the database where you intend to use the stored procedure.

Register the stored procedure

You can register the stored procedure using R, or you can run the CREATE PROCEDURE statement in T-SQL.

- Using T-SQL. If you are more comfortable with T-SQL, open SQL Server Management Studio (or any other client that can run SQL DDL commands) and execute the CREATE PROCEDURE statement using the code prepared by the `StoredProcedure` function.
- Using R. While you are still in your R environment, you can use the `registerStoredProcedure` function in `sqlrutils` to register the stored procedure with the database.

For example, you could register the stored procedure `sp_rsampl` in the instance and database defined in `sqlConnStr`, by making this R call:

```
R
```

```
registerStoredProcedure(sp_rsampl, sqlConnStr)
```

ⓘ Important

Regardless of whether you use R or SQL, you must run the statement using an account that has permissions to create new database objects.

Run using SQL

After the stored procedure has been created, open a connection to the SQL database using any client that supports T-SQL, and pass values for any parameters required by

the stored procedure.

Run using R

Some additional preparation is needed if you want to execute the stored procedure from R code, rather than from SQL Server. For example, if the stored procedure requires input values, you must set those input parameters before the function can be executed, and then pass those objects to the stored procedure in your R code.

The overall process of calling the prepared SQL stored procedure is as follows:

1. Call `getInputParameters` to get a list of input parameter objects.
2. Define a `$query` or set a `$value` for each input parameter.
3. Use `executeStoredProcedure` to execute the stored procedure from the R development environment, passing the list of input parameter objects that you set.

Example

This example shows the before and after versions of an R script that gets data from a SQL Server database, performs some transformations on the data, and saves it to a different database.

This simple example is used only to demonstrate how you might rearrange your R code to make it easier to convert to a stored procedure.

Before code preparation

R

```
sqlConnFrom <- "Driver={ODBC Driver 13 for SQL
Server};Server=MyServer01;Database=AirlineSrc;Trusted_Connection=Yes;"

sqlConnTo <- "Driver={ODBC Driver 13 for SQL
Server};Server=MyServer01;Database=AirlineTest;Trusted_Connection=Yes;"

sqlQueryAirline <- "SELECT TOP 10000 ArrDelay, CRSDepTime, DayOfWeek FROM
[AirlineDemoSmall]"

dsSqlFrom <- RxSqlServerData(sqlQuery = sqlQueryAirline, connectionString =
sqlConnFrom)

dsSqlTo <- RxSqlServerData(table = "cleanData", connectionString =
sqlConnTo)

xFunc <- function(data) {
```

```

    data$CRSDepHour <- as.integer(trunc(data$CRSDepTime))
    return(data)
  }

xVars <- c("CRSDepTime")

sqlCompute <- RxInSqlServer(numTasks = 4, connectionString = sqlConnTo)

rxOpen(dsSqlFrom)
rxOpen(dsSqlTo)

if (rxSqlServerTableExists("cleanData", connectionString = sqlConnTo)) {
  rxSqlServerDropTable("cleanData")}

rxDataStep(inData = dsSqlFrom,
           outFile = dsSqlTo,
           transformFunc = xFunc,
           transformVars = xVars,
           overwrite = TRUE)

```

ⓘ Note

When you use an ODBC connection rather than invoking the *RxSqlServerData* function, you must open the connection using *rxOpen* before you can perform operations on the database.

After code preparation

In the updated version, the first line defines the function name. All other code from the original R solution becomes a part of that function.

```

R

myetl1function <- function() {
  sqlConnFrom <- "Driver={ODBC Driver 13 for SQL
Server};Server=MyServer01;Database=Airline01;Trusted_Connection=Yes;"
  sqlConnTo <- "Driver={ODBC Driver 13 for SQL
Server};Server=MyServer02;Database=Airline02;Trusted_Connection=Yes;"

  sqlQueryAirline <- "SELECT TOP 10000 ArrDelay, CRSDepTime, DayOfWeek FROM
[AirlineDemoSmall]"

  dsSqlFrom <- RxSqlServerData(sqlQuery = sqlQueryAirline, connectionString
= sqlConnFrom)

  dsSqlTo <- RxSqlServerData(table = "cleanData", connectionString =
sqlConnTo)

  xFunc <- function(data) {

```

```
data$CRSDepHour <- as.integer(trunc(data$CRSDepTime))
return(data)}

xVars <- c("CRSDepTime")

sqlCompute <- RxInSqlServer(numTasks = 4, connectionString = sqlConnTo)

if (rxSqlServerTableExists("cleanData", connectionString = sqlConnTo))
{rxSqlServerDropTable("cleanData")}

rxDataStep(inData = dsSqlFrom,
           outFile = dsSqlTo,
           transformFunc = xFunc,
           transformVars = xVars,
           overwrite = TRUE)
return(NULL)
}
```

ⓘ Note

Although you do not need to open the ODBC connection explicitly as part of your code, an ODBC connection is still required to use `sqlrutils`.

See also

[sqlrutils reference](#)

executeStoredProcedure: Execute a SQL Stored Procedure

Article • 02/28/2023

`executeStoredProcedure`: Executes a stored procedure registered with the database

Usage

```
executeStoredProcedure(sqlSP, ..., connectionString = NULL)
```

Arguments

`sqlSP`

a valid StoredProcedure Object

...

Optional input and output parameters for the stored procedure. All of the parameters that do not have default queries or values assigned to them must be provided

`connectionString`

A character string (must be provided if the StoredProcedure object was created without a connection string). This function requires using an ODBC driver which supports ODBC 3.8 functionality.

`verbose`

Boolean. Whether to print out the command used to execute the stored procedure

Value

TRUE on success, FALSE on failure

Notes

This function relies that the ODBC driver used supports ODBC 3.8 features. Otherwise it will fail.

Examples

```
## Not run:

# See ?StoredProcedure for creating the "cleandata" table.

##### Example 1 #####
# Create a linear model and store in the "rdata" table.
train <- function(in_df) {
  factorLevels <-
c("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday")
  in_df[, "DayOfWeek"] <- factor(in_df[, "DayOfWeek"], levels=factorLevels)
  # The model formula
  formula <- ArrDelay ~ CRSDepTime + DayOfWeek + CRSDepHour:DayOfWeek

  # Train the model
  mm <- rxLinMod(formula, data = in_df, transformFunc = NULL, transformVars
= NULL)

  # Store the model into the database
  # rdata needs to be created beforehand
  conStr <- paste0("Driver={ODBC Driver 13 for SQL Server};Server=.;",
                  "Database=RevoTestDB;Trusted_Connection=Yes;")
  out.table = "rdata"
  # write the model to the table
  ds = RxOdbcData(table = out.table, connectionString = conStr)

  rxWriteObject(ds, "linmod.v1", mm, keyName = "key",
               valueName = "value")
  return(NULL)
}

conStr <- "Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;Trusted_Connection=Yes;"
# create an InputData object for the input data frame in_df
indata <- InputData("in_df",
                   defaultQuery = paste0("select top 10000
ArrDelay,CRSDepTime,",
                                         "DayOfWeek,CRSDepHour from
cleanData"))
# create the sql server stored procedure object
trainSP1 <- StoredProcedure('train', "spTrain_df_to_df", indata,
                           dbName = "RevoTestDB",
```

```

        connectionString = conStr,
        filePath = ".")
# spRegisterSp and executeStoredProcedure do not require a connection
string since we
# provided one when we created trainSP1
registerStoredProcedure(trainSP1)
executeStoredProcedure(trainSP1, verbose = TRUE)

##### Example 2 #####
# score1 makes a batch prediction given clean data(indata),
# model object(model_param), and the new name of the variable
# that is being predicted
score1 <- function(in_df, model_param, predVarNameInParam) {
  factorLevels <-
c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")
  in_df[, "DayOfWeek"] <- factor(in_df[, "DayOfWeek"], levels=factorLevels)
  mm <- rxReadObject(as.raw(model_param))
  # Predict
  result <- rxPredict(modelObject = mm,
                      data = in_df,
                      outData = NULL,
                      predVarNames = predVarNameInParam,
                      extraVarsToWrite = c("ArrDelay"),
                      writeModelVars = TRUE,
                      overwrite = TRUE)
  return(list(result = result, pvOutParam = mm$f.pvalue))
}

# create an InputData object for the input data frame in_df
indata <- InputData(name = "in_df", defaultQuery = "SELECT top 10 * from
cleanData")
# create InputParameter objects for model_param and predVarNameInParam
model <- InputParameter("model_param", "raw",
                      defaultQuery = paste("select top 1 value from rdata",
                                           "where [key] = 'linmod.v1'"))
predVarNameInParam <- InputParameter("predVarNameInParam", "character")
# create OutputData object for the data frame inside the return list
outData <- OutputData("result")
# create OutputParameter object for non data frame variable inside the
return list
pvOutParam <- OutputParameter("pvOutParam", "numeric")
scoreSP1 <- StoredProcedure(score1, "spScore_df_param_df", indata, model,
predVarNameInParam, outData, pvOutParam,
                          filePath = ".")
conStr <- "Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;Trusted_Connection=Yes;"
# connection string necessary for registrations and execution
# since we did not pass it to StoredProcedure
registerStoredProcedure(scoreSP1, conStr)
model <- executeStoredProcedure(scoreSP1, predVarNameInParam =
"ArrDelayEstimate", connectionString = conStr, verbose = TRUE)
model$data
model$params[[1]]
## End(Not run)

```


getInputParameters: Get a List of Input Parameters of a SQL Stored Procedure

Article • 02/28/2023

`getInputParameters`: returns a list of SQL Server parameter objects that describe the input parameters associated with a stored procedure

Usage

```
getInputParameters(sqlSP)
```

Arguments

sqlSP

A valid StoredProcedure Object

Value

A named list of SQL Server parameter objects (InputData, InputParameter) associated with the provided StoredProcedure object. The names are the names of the variables from the R function provided into StoredProcedure associated with the objects

Examples

```
## Not run:  
  
# See ?StoredProcedure for creating the `cleandata` table.  
# and ?executeStoredProcedure for creating the `rdata` table.  
  
# score1 makes a batch prediction given clean data(indata),  
# model object(model_param), and the new name of the variable  
# that is being predicted
```



```

score1 <- function(indata, model_param, predVarName) {
  indata[, "DayOfWeek"] <- factor(indata[, "DayOfWeek"],
levels=c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"))
  # The connection string
  conStr <- paste("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                  "Trusted_Connection=Yes;", sep = "")
  # The compute context
  computeContext <- RxInSqlServer(numTasks=4, connectionString=conStr)
  mm <- rxReadObject(as.raw(model_param))
  # Predict
  result <- rxPredict(modelObject = mm,
                      data = indata,
                      outData = NULL,
                      predVarNames = predVarName,
                      extraVarsToWrite = c("ArrDelay"),
                      writeModelVars = TRUE,
                      overwrite = TRUE)
}
# connections string
conStr <- paste0("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                 "Trusted_Connection=Yes;")
# create InputData Object for an input parameter that is a data frame
id <- InputData(name = "indata", defaultQuery = "SELECT * from cleanData")
# InputParameter for the model_param input variable
model <- InputParameter("model_param", "raw",
                        defaultQuery =
                          "select top 1 value from rdata where [key] =
'linmod.v1'")
# InputParameter for the predVarName variable
name <- InputParameter("predVarName", "character")
sp_df_df <- StoredProcedure(score1, "sTest", id, model, name,
                            filePath = ".")

# inspect the input parameters
getInputParameters(sp_df_df) # "model_param" "predVarName" "indata"

# register the stored procedure with a database
registerStoredProcedure(sp_df_df, conStr)
# assign a different query to the InputData so that it only uses the first
10 rows
id <- setInputDataQuery(id, "SELECT top 10 * from cleanData")
# assign a value to the name parameter
name <- setInputParameterValue(name, "ArrDelayEstimate")
# execute the stored procedure
model <- executeStoredProcedure(sp_df_df, id, name, connectionString =
conStr)
model$data
## End(Not run)

```

InputData: Input Data for SQL Stored Procedure: Class Generator

Article • 02/28/2023

InputData: generates an InputData Object that captures the information about the input parameter that is a data frame. The data frame needs to be populated upon the execution a given query. This object is necessary for creation of stored procedures in which the embedded R function takes in a data frame input parameter.

Usage

R

```
InputData(name, defaultQuery = NULL, query = NULL)
```

Arguments

name

A character string, the name of the data input parameter into the R function supplied to StoredProcedure.

defaultQuery

A character string specifying the default query that will retrieve the data if a different query is not provided at the time of the execution of the stored procedure. Must be a simple SELECT query.

query

A character string specifying the query that will be used to retrieve the data in the next run of the stored procedure.

Value

InputData Object

Examples

R

```
## Not run:

# See ?StoredProcedure for creating the "cleandata" table.

# train 1 takes a data frame with clean data and outputs a model
train1 <- function(in_df) {
  in_df[, "DayOfWeek"] <- factor(in_df[, "DayOfWeek"],
levels=c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"))
  # The model formula
  formula <- ArrDelay ~ CRSDepTime + DayOfWeek + CRSDepHour:DayOfWeek
  # Train the model
  rxSetComputeContext("local")
  mm <- rxLinMod(formula, data=in_df)
  mm <- rxSerializeModel(mm)

  return(list("mm" = mm))
}
# create InpuData Object for an input parameter that is a data frame
# note: if the input parameter is not a data frame use InputParameter
object
id <- InputData(name = "in_df",
                defaultQuery = paste0("select top 10000
ArrDelay,CRSDepTime,",
                                     "DayOfWeek,CRSDepHour from
cleanData"))
# create an OutputParameter object for the variable inside the return list
# note: if that variable is a data frame use OutputData object
out <- OutputParameter("mm", "raw")

# connections string
conStr <- paste0("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                "Trusted_Connection=Yes;")
# create the stored procedure object
sp_df_op <- StoredProcedure("train1", "spTest1", id, out,
                           filePath = ".")
# register the stored procedure with the database
registerStoredProcedure(sp_df_op, conStr)

# execute the stored procedure, note: non-data frame variables inside the
# return list are not returned to R. However, if the execution is not
sucessful
# the error will be displayed
model <- executeStoredProcedure(sp_df_op, connectionString = conStr)
# get the linear model
mm <- rxUnserializeModel(model$params$op1)
## End(Not run)
```


InputParameter: Input Parameter for SQL Stored Procedure: Class Generator

Article • 02/28/2023

InputParameter: generates an InputParameter Object, that captures the information about the input parameters of the R function that is to be embedded into a SQL Server Stored Procedure. Those will become the input parameters of the stored procedure. Supported R types of the input parameters are POSIXct, numeric, character, integer, logical, and raw.

Usage

```
InputParameter(name, type, defaultValue = NULL, defaultQuery = NULL,  
value = NULL, enableOutput = FALSE)
```

Arguments

name

A character string, the name of the input parameter object.

type

A character string representing the R type of the input parameter object.

defaultValue

Default value of the parameter. Not supported for "raw".

defaultQuery

A character string specifying the default query that will retrieve the data if a different query is not provided at the time of the execution of the stored procedure.

value

A value that will be used for the parameter in the next run of the stored procedure.

enableOutput

Make this an Input/Output Parameter

Value

InputParameter Object

Examples

```
## Not run:

# See ?StoredProcedure for creating the `cleandata` table.
# and ?executeStoredProcedure for creating the `rdata` table.

# score1 makes a batch prediction given clean data(indata),
# model object(model_param), and the new name of the variable
# that is being predicted
score1 <- function(indata, model_param, predVarName) {
  indata[, "DayOfWeek"] <- factor(indata[, "DayOfWeek"],
levels=c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"))
  # The connection string
  conStr <- paste("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                "Trusted_Connection=Yes;", sep = "")
  # The compute context
  computeContext <- RxInSqlServer(numTasks=4, connectionString=conStr)
  mm <- rxReadObject(as.raw(model_param))
  # Predict
  result <- rxPredict(modelObject = mm,
                      data = indata,
                      outData = NULL,
                      predVarNames = predVarName,
                      extraVarsToWrite = c("ArrDelay"),
                      writeModelVars = TRUE,
                      overwrite = TRUE)
}
# connections string
conStr <- paste0("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
```


OutputData: Output Data for SQL Stored Procedure: Class Generator

Article • 02/28/2023

`OutputData`: generates an `OutputData` Object that captures the information about the data frame that needs to be returned after the execution of the R function embedded into the stored procedure. This object must be created if the R function is returning a named list, where one of the items in the list is a data frame. The return list can contain at most one data frame.

Usage

```
OutputData(name)
```

Arguments

name

A character string, the name of the data frame variable.

Value

`OutputData` Object

Examples

```
## Not run:  
  
# See ?StoredProcedure for creating the "cleandata" table.  
  
# train 2 takes a data frame with clean data and outputs a model  
# as well as the data on the basis of which the model was built  
train2 <- function(in_df) {
```



```

in_df[,"DayOfWeek"] <- factor(in_df[,"DayOfWeek"],
levels=c("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"))
# The model formula
formula <- ArrDelay ~ CRSDepTime + DayOfWeek + CRSDepHour:DayOfWeek
# Train the model
rxSetComputeContext("local")
mm <- rxLinMod(formula, data=in_df, transformFunc=NULL,
transformVars=NULL)
mm <- rxSerializeModel(mm)
return(list(mm = mm, in_df = in_df))
}
# create InpuData Object for an input parameter that is a data frame
# note: if the input parameter is not a data frame use InputParameter
object
id <- InputData(name = "in_df",
                 defaultQuery = paste0("select top 10000
ArrDelay,CRSDepTime,",
                                     "DayOfWeek,CRSDepHour from
cleanData"))
out1 <- OutputData("in_df")
# create an OutputParameter object for the variable "mm" inside the return
list
out2 <- OutputParameter("mm", "raw")
# connections string
conStr <- paste0("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                 "Trusted_Connection=Yes;")
# create the stored procedure object
sp_df_op <- StoredProcedure(train2, "spTest2", id, out1, out2,
                           filePath = ".")

registerStoredProcedure(sp_df_op, conStr)
result <- executeStoredProcedure(sp_df_op, connectionString = conStr)
# Get back the linear model.
mm <- rxUnserializeModel(result$params$op1)
## End(Not run)

```

OutputParameter: Output Parameter for SQL Stored Procedure: Class Generator

Article • 02/28/2023

`OutputParameter`: generates an `OutputParameter` Object that captures the information about the output parameters of the function that is to be embedded into a SQL Server Stored Procedure. Those will become the output parameters of the stored procedure. Supported R types of the output parameters are `POSIXct`, `numeric`, `character`, `integer`, `logical`, and `raw`. This object must be created if the R function is returning a named list for non-data frame members of the list

Usage

```
OutputParameter(name, type)
```

Arguments

`name`

A character string, the name of the output parameter object.

`type`

R type of the output parameter object.

Value

`OutputParameter` Object

Examples

```

## Not run:

# See ?StoredProcedure for creating the "cleandata" table.

# train 2 takes a data frame with clean data and outputs a model
# as well as the data on the basis of which the model was built
train2 <- function(in_df) {
  in_df[,"DayOfWeek"] <- factor(in_df[,"DayOfWeek"],
levels=c("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"))
  # The model formula
  formula <- ArrDelay ~ CRSDepTime + DayOfWeek + CRSDepHour:DayOfWeek
  # Train the model
  rxSetComputeContext("local")
  mm <- rxLinMod(formula, data=in_df, transformFunc=NULL,
transformVars=NULL)
  mm <- rxSerializeModel(mm)
  return(list(mm = mm, in_df = in_df))
}
# create InputData Object for an input parameter that is a data frame
# note: if the input parameter is not a data frame use InputParameter
object
id <- InputData(name = "in_df",
                defaultQuery = paste0("select top 10000
ArrDelay,CRSDepTime,",
                                     "DayOfWeek,CRSDepHour from
cleanData"))
out1 <- OutputData("in_df")
# create an OutputParameter object for the variable "mm" inside the return
list
out2 <- OutputParameter("mm", "raw")
# connections string
conStr <- paste0("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                "Trusted_Connection=Yes;")
# create the stored procedure object
sp_df_op <- StoredProcedure(train2, "spTest2", id, out1, out2,
                           filePath = ".")
registerStoredProcedure(sp_df_op, conStr)
result <- executeStoredProcedure(sp_df_op, connectionString = conStr)
# Get back the linear model.
mm <- rxUnserializeModel(result$params$op1)
## End(Not run)

```

registerStoredProcedure: Register a SQL Stored Procedure with a Database

Article • 02/28/2023

`registerStoredProcedure`: Uses the `StoredProcedure` object to register the stored procedure with the specified database

Usage

```
registerStoredProcedure(sqlSP, connectionString = NULL)
```

Arguments

`sqlSP`

a valid `StoredProcedure` object

`connectionString`

A character string (must be provided if the `StoredProcedure` object was created without a connection string)

Value

TRUE on success, FALSE on failure

Examples

```
## Not run:  
  
# See ?StoredProcedure for creating the "cleandata" table.
```

```

# train 1 takes a data frame with clean data and outputs a model
train1 <- function(in_df) {
  in_df[, "DayOfWeek"] <- factor(in_df[, "DayOfWeek"],
levels=c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"))
  # The model formula
  formula <- ArrDelay ~ CRSDepTime + DayOfWeek + CRSDepHour:DayOfWeek
  # Train the model
  rxSetComputeContext("local")
  mm <- rxLinMod(formula, data=in_df)
  mm <- rxSerializeModel(mm)
  return(list("mm" = mm))
}
# create InputData Object for an input parameter that is a data frame
# note: if the input parameter is not a data frame use InputParameter object
id <- InputData(name = "in_df",
                defaultQuery = paste0("select top 10000 ArrDelay, CRSDepTime,",
                "DayOfWeek, CRSDepHour from cleanData"))

# create an OutputParameter object for the variable inside the return list
# note: if that variable is a data frame use OutputData object
out <- OutputParameter("mm", "raw")

# connections string
conStr <- paste0("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                "Trusted_Connection=Yes;")
# create the stored procedure object
sp_df_op <- StoredProcedure("train1", "spTest1", id, out,
                filePath = ".")
# register the stored procedure with the database
registerStoredProcedure(sp_df_op, conStr)
model <- executeStoredProcedure(sp_df_op, connectionString = conStr)

# Getting back the model by unserializing it.
mm <- rxUnserializeModel(model$params$op1)
## End(Not run)

```

setInputDataQuery: Assign a Query to the Input Data Parameter of the SQL Stored Procedure

Article • 02/28/2023

`setInputDataQuery`: assigns a query to the `InputData` parameter of the stored procedure that is going to populate the input data frame of the embedded R function in the next run of the stored procedure.

Usage

```
setInputDataQuery(inputData, query)
```

Arguments

`inputData`

A character string, the name of the data frame input parameter into the R function.

`query`

A character string representing a query.

Value

InputData Object

Examples

```
## Not run:
```

```

# See ?StoredProcedure for creating the `cleandata` table.
# and ?executeStoredProcedure for creating the `rdata` table.

# score1 makes a batch prediction given clean data(indata),
# model object(model_param), and the new name of the variable
# that is being predicted
score1 <- function(indata, model_param, predVarName) {
  indata[, "DayOfWeek"] <- factor(indata[, "DayOfWeek"],
levels=c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"))
  # The connection string
  conStr <- paste("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                  "Trusted_Connection=Yes;", sep = "")
  # The compute context
  computeContext <- RxInSqlServer(numTasks=4, connectionString=conStr)
  mm <- rxReadObject(as.raw(model_param))
  # Predict
  result <- rxPredict(modelObject = mm,
                      data = indata,
                      outData = NULL,
                      predVarNames = predVarName,
                      extraVarsToWrite = c("ArrDelay"),
                      writeModelVars = TRUE,
                      overwrite = TRUE)
}
# connections string
conStr <- paste0("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                 "Trusted_Connection=Yes;")
# create InpuData Object for an input parameter that is a data frame
id <- InputData(name = "indata", defaultQuery = "SELECT * from cleanData")
# InputParameter for the model_param input variable
model <- InputParameter("model_param", "raw",
                        defaultQuery =
                          "select top 1 value from rdata where [key] =
'linmod.v1'")
# InputParameter for the predVarName variable
name <- InputParameter("predVarName", "character")
sp_df_df <- StoredProcedure(score1, "sTest", id, model, name,
                           filePath = ".")
# register the stored procedure with a database
registerStoredProcedure(sp_df_df, conStr)
# assign a different query to the InputData so that it only uses the first
10 rows
id <- setInputDataQuery(id, "SELECT top 10 * from cleanData")
# assign a value to the name parameter
name <- setInputParameterValue(name, "ArrDelayEstimate")
# execute the stored procedure
model <- executeStoredProcedure(sp_df_df, id, name, connectionString =
conStr)
model$data
## End(Not run)

```

setInputParameterValue: Assign a Value to the Input Data Parameter of the SQL Stored Procedure

Article • 02/28/2023

`setInputParameterValue`: assigns a value to an input parameter of the stored procedure/embedded R function that is going to be used in the next run of the stored procedure.

Usage

```
setInputParameterValue(inParam, value)
```

Arguments

`inParam`

A character string, the name of input parameter into the R function.

`value`

A value that is to be bound to the input parameter. Note: binding for input parameters of type "raw" is not supported.

Value

InputParameter Object

Examples

```
## Not run:
```



```

# See ?StoredProcedure for creating the `cleandata` table.
# and ?executeStoredProcedure for creating the `rdata` table.

# score1 makes a batch prediction given clean data(indata),
# model object(model_param), and the new name of the variable
# that is being predicted
score1 <- function(indata, model_param, predVarName) {
  indata[, "DayOfWeek"] <- factor(indata[, "DayOfWeek"],
levels=c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"))
  # The connection string
  conStr <- paste("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                  "Trusted_Connection=Yes;", sep = "")
  # The compute context
  computeContext <- RxInSqlServer(numTasks=4, connectionString=conStr)
  mm <- rxReadObject(as.raw(model_param))
  # Predict
  result <- rxPredict(modelObject = mm,
                      data = indata,
                      outData = NULL,
                      predVarNames = predVarName,
                      extraVarsToWrite = c("ArrDelay"),
                      writeModelVars = TRUE,
                      overwrite = TRUE)
}
# connections string
conStr <- paste0("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                 "Trusted_Connection=Yes;")
# create InpuData Object for an input parameter that is a data frame
id <- InputData(name = "indata", defaultQuery = "SELECT * from cleanData")
# InputParameter for the model_param input variable
model <- InputParameter("model_param", "raw",
                        defaultQuery =
                          "select top 1 value from rdata where [key] =
'linmod.v1'")
# InputParameter for the predVarName variable
name <- InputParameter("predVarName", "character")
sp_df_df <- StoredProcedure(score1, "sTest", id, model, name,
                            filePath = ".")
# register the stored procedure with a database
registerStoredProcedure(sp_df_df, conStr)
# assign a different query to the InputData so that it only uses the first
10 rows
id <- setInputDataQuery(id, "SELECT top 10 * from cleanData")
# assign a value to the name parameter
name <- setInputParameterValue(name, "ArrDelayEstimate")
# execute the stored procedure
model <- executeStoredProcedure(sp_df_df, id, name, connectionString =
conStr)
model$data
## End(Not run)

```

StoredProcedure: SQL Server Stored Procedure: Class Generator

Article • 02/28/2023

StoredProcedure: generates a SQLServer Stored Procedure Object and optionally a .sql file containing a query to create a stored procedure. `StoredProcedure$registrationVec` contains strings representing the queries needed for creation of the stored procedure

Usage

```
StoredProcedure (func, spName, ..., filePath = NULL ,dbName = NULL,  
connectionString = NULL, batchSeparator = "GO")
```

Arguments

func

A valid R function or a string name of a valid R function: 1) All of the variables that the function relies on should be defined either inside the function or come in as input parameters. Among the input parameters there can be at most 1 data frame 2) The function should return either a data frame, a named list, or NULL. There can be at most one data frame inside the list.

spName

A character string specifying name for the stored procedure.

...

Optional input and output parameters for the stored procedure; must be objects of classes `InputData`, `InputParameter`, or `outputParameter`.

filePath

A character string specifying a path to the directory in which to create the .sql. If NULL the .sql file is not generated.

dbName

A character string specifying name of the database to use.

connectionString

A character string specifying the connection string.

batchSeparator

Desired SQL batch separator (only relevant if filePath is defined)

Value

SQLServer Stored Procedure Object

Examples

```
## Not run:

##### Example 1 #####
# etl1 - reads from and write directly to the database
etl1 <- function() {
  # The query to get the data
  qq <- "select top 10000 ArrDelay,CRSDepTime,DayOfWeek from
AirlineDemoSmall"
  # The connection string
  conStr <- paste("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
                 "Trusted_Connection=Yes;", sep = "")
  # The data source - retrieves the data from the database
  dsSqs1 <- RxSqlServerData(sqlQuery=qq, connectionString=conStr)
  # The destination data source
  dsSqs2 <- RxSqlServerData(table = "cleanData", connectionString =
conStr)
  # A transformation function
  transformFunc <- function(data) {
    data$CRSDepHour <- as.integer(trunc(data$CRSDepTime))
    return(data)
  }
}
```

```

# The transformation variables
transformVars <- c("CRSDepTime")
rxDataStep(inData = dsSqls,
           outFile = dsSqls2,
           transformFunc=transformFunc,
           transformVars=transformVars,
           overwrite = TRUE)
  return(NULL)
}
# Create a StoredProcedure object
sp_ds_ds <- StoredProcedure(etl1, "spTest",
                          filePath = ".", dbName = "RevoTestDB")
# Define a connection string
conStr <- paste("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
               "Trusted_Connection=Yes;", sep = "")
# register the stored procedure with a database
registerStoredProcedure(sp_ds_ds, conStr)
# execute the stored procedure
executeStoredProcedure(sp_ds_ds, connectionString = conStr)

##### Example 2 #####
# train 1 takes a data frame with clean data and outputs a model
train1 <- function(in_df) {
  in_df[, "DayOfWeek"] <- factor(in_df[, "DayOfWeek"],
levels=c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"))
  # The model formula
  formula <- ArrDelay ~ CRSDepTime + DayOfWeek + CRSDepHour:DayOfWeek
  # Train the model
  rxSetComputeContext("local")
  mm <- rxLinMod(formula, data=in_df)
  mm <- rxSerializeModel(mm)
  return(list("mm" = mm))
}
# create InpuData Object for an input parameter that is a data frame
# note: if the input parameter is not a data frame use InputParameter
object
id <- InputData(name = "in_df",
                defaultQuery = paste0("select top 10000
ArrDelay,CRSDepTime,",
                                     "DayOfWeek,CRSDepHour from
cleanData"))
# create an OutputParameter object for the variable inside the return list
# note: if that variable is a data frame use OutputData object
out <- OutputParameter("mm", "raw")

# connections string
conStr <- paste0("Driver={ODBC Driver 13 for SQL
Server};Server=.;Database=RevoTestDB;",
               "Trusted_Connection=Yes;")
# create the stored procedure object
sp_df_op <- StoredProcedure("train1", "spTest1", id, out,
                          filePath = ".")


```

```
# register the stored procedure with the database
registerStoredProcedure(sp_df_op, conStr)


# get the linear model
model <- executeStoredProcedure(sp_df_op, connectionString = conStr)
mm <- rxUnserializeModel(model$params$op1)
## End(Not run)
```

SQL Server 2017 Release Notes





Article • 03/31/2023

Applies to:  SQL Server 2017 (14.x) and later

This article describes limitations and issues with SQL Server 2017. For related information, see:

- [What's New in SQL Server 2017](#)
- [SQL Server on Linux release notes](#)
- [SQL Server 2017 Cumulative updates](#)  for information about the latest cumulative update (CU) release

Try SQL Server!

-  [Download SQL Server 2017](#) 
-  [Spin up a Virtual Machine with SQL Server 2017](#) 

Note

SQL Server 2019 preview is now available. For more information, see [What's New in SQL Server 2019](#).

SQL Server 2017 - general availability release (October 2017)

Database Engine

- **Issue and customer impact:** After upgrade, the existing FILESTREAM network share may be no longer available.
- **Workaround:** First, reboot the computer and check if the FILESTREAM network share is available. If the share is still not available, complete the following steps:
 1. In SQL Server Configuration Manager, right-click the SQL Server instance, and click **Properties**.
 2. In the **FILESTREAM** tab clear **Enable FILESTREAM for file I/O streaming access**, then click **Apply**.
 3. Check **Enable FILESTREAM for file I/O streaming access** again with the original share name and click **Apply**.

Master Data Services (MDS)

- **Issue and customer impact:** On the user permissions page, when granting permission to the root level in the entity tree view, you see the following error:
`"The model permission cannot be saved. The object guid is not valid"`
- **Workaround:**
 - Grant permission on the sub nodes in the tree view instead of the root level.

Analysis Services

- **Issue and customer impact:** Data connectors for the following sources are not yet available for tabular models at the 1400 compatibility level.
 - Amazon Redshift
 - IBM Netezza
 - Impala
- **Workaround:** None.
- **Issue and customer impact:** Direct Query models at the 1400 compatibility level with perspectives can fail on querying or discovering metadata.
- **Workaround:** Remove perspectives and redeploy.

Tools

- **Issue and customer impact:** Running *DReplay* fails with the following message:
"Error DReplay Unexpected error occurred!".
- **Workaround:** None.

SQL Server 2017 Release Candidate (RC2 - August 2017)

There are no release notes for SQL Server on Windows related to this release. See [SQL Server on Linux Release notes](#).

SQL Server 2017 Release Candidate (RC1 - July 2017)

SQL Server Integration Services (SSIS) (RC1 - July 2017)

- **Issue and customer impact:** The parameter *runincluster* of the stored procedure [catalog].[create_execution] is renamed to *runinscaleout* for consistency and readability.
- **Work around:** If you have existing scripts to run packages in Scale Out, you have to change the parameter name from *runincluster* to *runinscaleout* to make the scripts work in RC1.
- **Issue and customer impact:** SQL Server Management Studio (SSMS) 17.1 and earlier versions can't trigger package execution in Scale Out in RC1. The error message is: "@runincluster is not a parameter for procedure create_execution." This issue is fixed in the next release of SSMS, version 17.2. Versions 17.2 and later of SSMS support the new parameter name and package execution in Scale Out.
- **Work around:** Until SSMS version 17.2 is available:
 1. Use your existing version of SSMS to generate the package execution script.
 2. Change the name of the *runincluster* parameter to *runinscaleout* in the script.
 3. Run the script.

SQL Server 2017 CTP 2.1 (May 2017)

Documentation (CTP 2.1)

- **Issue and customer impact:** Documentation for SQL Server 2017 (14.x) is limited and content is included with the SQL Server 2016 (13.x) documentation set. Content in articles that is specific to SQL Server 2017 (14.x) is noted with **Applies To**.
- **Issue and customer impact:** No offline content is available for SQL Server 2017 (14.x).

SQL Server Reporting Services (CTP 2.1)

- **Issue and customer impact:** If you have both SQL Server Reporting Services and Power BI Report Server on the same machine and uninstall one of them, you cannot connect to the remaining report server with Report Server Configuration Manager.
- **Work around** To work around this issue, you must perform the following operations after uninstalling one of the servers.

1. Launch a command prompt in Administrator mode.
2. Go to the directory where the remaining report server is installed.

Default location for Power BI Report Server: C:\Program Files\Microsoft Power BI Report Server

Default location for SQL Server Reporting Services: C:\Program Files\Microsoft SQL Server Reporting Services

3. Then go to the next folder, which is either *SSRS* or *PBIRS* depending on what is remaining.
4. Go to the WMI folder.
5. Run the following command:

Console

```
regsvr32 /i ReportingServicesWMIProvider.dll
```

If you see the following error, ignore it.

```
The module "ReportingServicesWMIProvider.dll" was loaded but the entry-point DLLInstall was not found. Make sure that "ReportingServicesWMIProvider.dll" is a valid DLL or OCX file and then try again.
```

TSqlLanguageService.msi (CTP 2.1)

- **Issue and customer impact:** After installing on a computer that has a 2016 version of *TSqlLanguageService.msi* installed (either through SQL Setup or as a standalone redistributable) the v13.* (SQL 2016) versions of *Microsoft.SqlServer.Management.SqlParser.dll* and *Microsoft.SqlServer.Management.SystemMetadataProvider.dll* are removed. Any

application that has a dependency on the 2016 versions of those assemblies stops working and generate an error similar to: *error : Could not load file or assembly 'Microsoft.SqlServer.Management.SqlParser, Version=13.0.0.0, Culture=neutral, PublicKeyToken=89845dcd8080cc91' or one of its dependencies. The system cannot find the file specified.*

In addition, attempts to reinstall a 2016 version of `TSqlLanguageService.msi` fail with the message: *Installation of Microsoft SQL Server 2016 T-SQL Language Service failed because a higher version already exists on the machine.*

- **Workaround** To work around this issue and fix an application that depends on the v13 version of the assemblies follow these steps:
 1. Go to **Add/Remove Programs**
 2. Find *Microsoft SQL Server 2019 T-SQL Language Service CTP2.1*, right-click it, and select **Uninstall**.
 3. After the component is removed, repair the application that is broken or reinstall the appropriate version of `TSqlLanguageService.MSI`.

This workaround removes the v14 version of those assemblies, so any applications that depend on the v14 versions will no longer function. If those assemblies are needed, then a separate installation without any side-by-side 2016 installs is required.

SQL Server 2017 CTP 2.0 (April 2017)

Documentation (CTP 2.0)

- **Issue and customer impact:** Documentation for SQL Server 2017 (14.x) is limited and content is included with the SQL Server 2016 (13.x) documentation set. Content in articles that is specific to SQL Server 2017 (14.x) is noted with **Applies To**.
- **Issue and customer impact:** No offline content is available for SQL Server 2017 (14.x).

Always On availability groups

- **Issue and customer impact:** A SQL Server instance hosting an availability group secondary replica crashes if the SQL Server major version is lower than the instance

that hosts the primary replica. Affects upgrades from all supported versions of SQL Server that host availability groups to SQL Server SQL Server 2017 (14.x) CTP 2.0. The issue occurs under the following conditions.

1. User upgrades SQL Server instance hosting secondary replica in accordance with [best practices](#).
 2. After upgrade, a failover occurs and a newly upgraded secondary becomes primary before completing upgrade for all secondary replicas in the availability group. The old primary is now a secondary, which is lower version than primary.
 3. The availability group is in an unsupported configuration and any remaining secondary replicas might be vulnerable to crash.
- **Workaround** Connect to the SQL Server instance that hosts the new primary replica, and remove the faulty secondary replica from the configuration.

```
SQL
```








```
ALTER AVAILABILITY GROUP agName REMOVE REPLICA ON NODE instanceName;
```

The instance of SQL Server that hosted the secondary replica recovers.

More information

- [SQL Server Reporting Services release notes](#).
- [Known Issues for Machine Learning Services](#)
- [SQL Server Update Center - links and information for all supported versions](#)

Get help

- [Ideas for SQL: Have suggestions for improving SQL Server?](#) 
- [Microsoft Q & A \(SQL Server\)](#)
- [DBA Stack Exchange \(tag sql-server\): Ask SQL Server questions](#) 
- [Stack Overflow \(tag sql-server\): Answers to SQL development questions](#) 
- [Reddit: General discussion about SQL Server](#) 
- [Microsoft SQL Server License Terms and Information](#) 
- [Support options for business users](#) 
- [Contact Microsoft](#) 
- [Additional SQL Server help and feedback](#)

Contribute to SQL documentation

Did you know that you can edit SQL content yourself? If you do so, not only do you help improve our documentation, but you also get credited as a contributor to the page.

For more information, see [How to contribute to SQL Server documentation](#)



Known issues for Python and R in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Important

The support for Machine Learning Server (previously known as R Server) ended on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This article describes known problems or limitations with the Python and R components that are provided in [SQL Server Machine Learning Services](#) and [SQL Server 2016 R Services](#).

Setup and configuration issues

For a description of processes related to initial setup and configuration, see [Install SQL Server Machine Learning Services](#). It contains information about upgrades, side-by-side installation, and installation of new R or Python components.

Inconsistent results in MKL computations due to missing environment variable

Applies to: R_SERVER binaries 9.0, 9.1, 9.2 or 9.3.

R_SERVER uses the Intel Math Kernel Library (MKL). For computations involving MKL, inconsistent results can occur if your system is missing an environment variable.

Set the environment variable `'MKL_CBWR'=AUTO` to ensure conditional numerical reproducibility in R_SERVER. For more information, see [Introduction to Conditional Numerical Reproducibility \(CNR\)](#) [↗](#).

Workaround

1. In Control Panel, select **System and Security** > **System** > **Advanced System Settings** > **Environment Variables**.

2. Create a new User or System variable.

- Set Variable to `MKL_CBWR`.
- Set the Value to `AUTO`.

3. Restart R_SERVER. On SQL Server, you can restart SQL Server Launchpad Service.

ⓘ Note

If you are running the SQL Server 2019 (15.x) on Linux, edit or create `.bash_profile` in your user home directory, adding the line `export MKL_CBWR="AUTO"`. Execute this file by typing `source .bash_profile` at a bash command prompt. Restart R_SERVER by typing `sys.getenv()` at the R command prompt.

R Script runtime error (SQL Server 2017 CU 5 - CU 7 regression)

For SQL Server 2017 (14.x), in cumulative updates 5 through 7, there is a regression in the `rlauncher.config` file where the temp directory file path includes a space. This regression is corrected in CU 8.

The error you will see when running R script includes the following messages:

Unable to communicate with the runtime for 'R' script. Please check the requirements of 'R' runtime.

STDERR message(s) from external script:

Fatal error: cannot create 'R_TempDir'

Workaround

Apply CU 8 when it becomes available. Alternatively, you can recreate `rlauncher.config` by running `registerrext` with `uninstall/install` on an elevated command prompt.

Windows Command Prompt

```
<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRExt.exe  
/uninstall /sqlbinpath:<SQLInstanceBinnPath> /userpoolsize:0 /instance:  
<SQLInstanceName>
```

```
<SQLInstancePath>\R_SERVICES\library\RevoScaleR\rxLibs\x64\RegisterRExt.exe
```

```
/install /sqlbinnpath:<SQLInstanceBinnPath> /userpoolsize:0 /instance:  
<SQLInstanceName>
```

The following example shows the commands with the default instance

"MSSQL14.MSSQLSERVER" installed into `C:\Program Files\Microsoft SQL Server\`:

Windows Command Prompt

```
"C:\Program Files\Microsoft SQL  
Server\MSSQL14.MSSQLSERVER\R_SERVICES\library\RevoScaleR\rxLibs\x64\Register  
Rext.exe" /uninstall /sqlbinnpath:"C:\Program Files\Microsoft SQL  
Server\MSSQL14.MSSQLSERVER\MSSQL\Binn" /userpoolsize:0 /instance:MSSQLSERVER
```

```
"C:\Program Files\Microsoft SQL  
Server\MSSQL14.MSSQLSERVER\R_SERVICES\library\RevoScaleR\rxLibs\x64\Register  
Rext.exe" /install /sqlbinnpath:"C:\Program Files\Microsoft SQL  
Server\MSSQL14.MSSQLSERVER\MSSQL\Binn" /userpoolsize:0 /instance:MSSQLSERVER
```

Unable to install SQL Server machine learning features on a domain controller

If you try to install SQL Server 2016 (13.x) R Services or SQL Server Machine Learning Services on a domain controller, setup fails, with these errors:

An error occurred during the setup process of the feature

Cannot find group with identity

Component error code: 0x80131509

The failure occurs because, on a domain controller, the service can't create the 20 local accounts required to run machine learning. In general, we don't recommend installing SQL Server on a domain controller. For more information, see [Support bulletin 2032911](#).

Install the latest service release to ensure compatibility with Microsoft R Client

If you install the latest version of Microsoft R Client and use it to run R on SQL Server in a remote compute context, you might get an error like the following:

You are running version 9.x.x of Microsoft R Client on your computer, which is incompatible with Microsoft R Server version 8.x.x. Download and install a compatible

version.

SQL Server 2016 (13.x) requires that the R libraries on the client exactly match the R libraries on the server. The restriction has been removed for releases later than R Server 9.0.1. However, if you encounter this error, verify the version of the R libraries that's used by your client and the server and, if necessary, update the client to match the server version.

The version of R that is installed with SQL Server R Services is updated whenever a SQL Server service release is installed. To ensure that you always have the most up-to-date versions of R components, be sure to install all service packs.

To ensure compatibility with Microsoft R Client 9.0.0, install the updates that are described in this archived version of support article [KB3210262](#).

To avoid problems with R packages, you can also upgrade the version of the R libraries that are installed on the server, by changing your servicing agreement to use the Modern Lifecycle Support policy, as described in [the next section](#). When you do so, the version of R that's installed with SQL Server is updated on the same schedule used for updates of Machine Learning Server (formerly Microsoft R Server).

Applies to: SQL Server 2016 (13.x) R Services, with R Server version 9.0.0 or earlier

R components missing from SQL Server 2017 CU 3 setup

A limited number of Azure virtual machines were provisioned without the R installation files that should be included with SQL Server. The issue applies to virtual machines provisioned in the period from 2018-01-05 to 2018-01-23. This issue might also affect on-premises installations, if you applied the CU 3 update for SQL Server 2017 (14.x) during the period from 2018-01-05 to 2018-01-23.

A service release has been provided that includes the correct version of the R installation files.

- [Cumulative Update Package 3 for SQL Server 2017 KB4052987](#).

To install the components and repair SQL Server 2017 (14.x) CU 3, you must uninstall CU 3, and reinstall the updated version:

1. Download the updated CU 3 installation file, which includes the R installers.
2. Uninstall CU 3. In Control Panel, search for **Uninstall an update**, and then select "Hotfix 3015 for SQL Server 2017 (KB4052987) (64-bit)". Proceed with uninstall steps.

3. Reinstall the CU 3 update, by double-clicking on the update for KB4052987 that you downloaded: `SQLServer2017-KB4052987-x64.exe`. Follow the installation instructions.

Unable to install Python components in offline installations of SQL Server 2017 or later

If you install a pre-release version of SQL Server 2017 (14.x) on a computer without internet access, the installer might fail to display the page that prompts for the location of the downloaded Python components. In such an instance, you can install the Machine Learning Services feature, but not the Python components.

This issue is fixed in the release version. Also, this limitation doesn't apply to R components.

Applies to: SQL Server 2017 (14.x) with Python

Warn of incompatible version when you connect to an older version of SQL Server R Services from a client by using SQL Server 2017

When you run R code in a SQL Server 2016 (13.x) compute context, you might see the following error:

You are running version 9.0.0 of Microsoft R Client on your computer, which is incompatible with the Microsoft R Server version 8.0.3. Download and install a compatible version.

This message is displayed if either of the following two statements is true:

- You installed R Server (Standalone) on a client computer by using the setup wizard for SQL Server 2017 (14.x).
- You installed Microsoft R Server by using the [separate Windows installer](#).

To ensure that the server and client use the same version you might need to use *binding*, supported for Microsoft R Server 9.0 and later releases, to upgrade the R components in SQL Server 2016 (13.x) instances. To determine if support for upgrades is available for your version of R Services, see [Upgrade an instance of R Services using SqlBindR.exe](#).

Applies to: SQL Server 2016 (13.x) R Services, with R Server version 9.0.0 or earlier

Setup for SQL Server 2016 service releases might fail to install newer versions of R components

When you install a cumulative update or install a service pack for SQL Server 2016 (13.x) on a computer that isn't connected to the internet, the setup wizard might fail to display the prompt that lets you update the R components by using downloaded CAB files. This failure typically occurs when multiple components were installed together with the database engine.

As a workaround, you can install the service release by using the command line and specifying the `MRCACHEDIRECTORY` argument as shown in this example, which installs CU 1 updates:

```
C:\<path to installation media>\SQLServer2016-KB3164674-x64.exe /Action=Patch /IACCEPTROPENLICENSETERMS /MRCACHEDIRECTORY=<path to CU 1 CAB files>
```

To get the latest installers, see [Install machine learning components without internet access](#).

Applies to: SQL Server 2016 (13.x) R Services, with R Server version 9.0.0 or earlier

Launchpad services fails to start if the version is different from the R version

If you install SQL Server R Services separately from the database engine, and the build versions are different, you might see the following error in the System Event log:

The SQL Server Launchpad service failed to start due to the following error: The service did not respond to the start or control request in a timely fashion.

For example, this error might occur if you install the database engine by using the release version, apply a patch to upgrade the database engine, and then add the R Services feature by using the release version.

To avoid this problem, use a utility such as File Manager to compare the versions of Launchpad.exe with version of SQL binaries, such as `sqlkd.d11`. All components should have the same version number. If you upgrade one component, be sure to apply the same upgrade to all other installed components.

Look for Launchpad in the `Binn` folder for the instance. For example, in a default installation of SQL Server 2016 (13.x), the path might be `C:\Program Files\Microsoft SQL Server\MSSQL.13.InstanceNameMSSQL\Binn`.

Remote compute contexts are blocked by a firewall in SQL Server instances that are running on Azure virtual machines

If you have installed SQL Server on an Azure virtual machine, you might not be able to use compute contexts that require the use of the virtual machine's workspace. The reason is that, by default, the firewall on Azure virtual machines includes a rule that blocks network access for local R user accounts.

As a workaround, on the Azure VM, open **Windows Firewall with Advanced Security**, select **Outbound Rules**, and disable the following rule: **Block network access for R local user accounts in SQL Server instance MSSQLSERVER**. You can also leave the rule enabled, but change the security property to **Allow if secure**.

Implied authentication in SQL Server 2016 Express edition

When you run R jobs from a remote data-science workstation by using Integrated Windows authentication, SQL Server uses *implied authentication* to generate any local ODBC calls that might be required by the script. However, this feature didn't work in the RTM build of SQL Server 2016 (13.x) Express edition.

To fix the issue, we recommend that you upgrade to a later service release. If upgrade isn't feasible, as a workaround, use a SQL login to run remote R jobs that might require embedded ODBC calls.

Applies to: SQL Server 2016 (13.x) R Services Express edition

Performance limits when libraries used by SQL Server are called from other tools

It is possible to call the machine learning libraries that are installed for SQL Server from an external application, such as RGui. Doing so might be the most convenient way to accomplish certain tasks, such as installing new packages, or running ad hoc tests on very short code samples. However, outside of SQL Server, performance might be limited.

For example, even if you are using the Enterprise edition of SQL Server, R runs in single-threaded mode when you run your R code by using external tools. To get the benefits of performance in SQL Server, initiate a SQL Server connection and use [sp_execute_external_script](#) to call the external script runtime.

In general, avoid calling the machine learning libraries that are used by SQL Server from external tools. If you need to debug R or Python code, it is typically easier to do so

outside of SQL Server. To get the same libraries that are in SQL Server, you can install Microsoft R Client or [SQL Server 2017 Machine Learning Server \(Standalone\)](#).

SQL Server Data Tools doesn't support permissions required by external scripts

When you use Visual Studio or SQL Server Data Tools to publish a database project, if any principal has permissions specific to external script execution, you might get an error like this one:

TSQL Model: Error detected when reverse engineering the database. The permission was not recognized and was not imported.

Currently the DACPAC model doesn't support the permissions used by R Services or Machine Learning Services, such as `GRANT ANY EXTERNAL SCRIPT`, or `EXECUTE ANY EXTERNAL SCRIPT`. This issue will be fixed in a later release.

As a workaround, run the additional `GRANT` statements in a post-deployment script.

External script execution is throttled due to resource governance default values

In Enterprise edition, you can use resource pools to manage external script processes. In some early release builds, the maximum memory that could be allocated to the R processes was 20 percent. Therefore, if the server had 32 GB of RAM, the R executables (`RTerm.exe` and `BxlServer.exe`) could use a maximum of 6.4 GB in a single request.

If you encounter resource limitations, check the current default. If 20 percent isn't enough, see the documentation for SQL Server on how to change this value.

Applies to: SQL Server 2016 (13.x) R Services, Enterprise edition

Error when using `sp_execute_external_script` without `libc++.so` on Linux

On a clean Linux machine that doesn't have `libc++.so` installed, running a `sp_execute_external_script` (SPEES) query with Java or an external language fails because `commonlauncher.so` fails to load `libc++.so`.

For example:

SQL

```
EXECUTE sp_execute_external_script @language = N'Java'  
    , @script = N'JavaTestPackage.PassThrough'  
    , @parallel = 0  
    , @input_data_1 = N'select 1'  
WITH RESULT SETS((col1 INT NOT NULL));  
GO
```

This fails with a message similar to the following:

text

Msg 39012, Level 16, State 14, Line 0

Unable to communicate with the runtime for 'Java' script for request id: 94257840-1704-45E8-83D2-2F74AEB46CF7. Please check the requirements of 'Java' runtime.

The `mssql-launchpadd` logs will show an error message similar to the following:

text

```
Oct 18 14:03:21 sqlxhtmls launchpadd[57471]: [launchpad] 2019/10/18 14:03:21  
WARNING: PopulateLauncher failed: Library /opt/mssql-  
extensibility/lib/commonlauncher.so not loaded. Error: libc++.so.1: cannot  
open shared object file: No such file or directory
```

Workaround

You can perform one of the following workarounds:

1. Copy `libc++*` from `/opt/mssql/lib` to the default system path `/lib64`
2. Add the following entries to `/var/opt/mssql/mssql.conf` to expose the path:

text

```
[extensibility]  
readabledirectories = /opt/mssql
```

Applies to: SQL Server 2019 (15.x) on Linux

Installation or upgrade error on FIPS enabled servers

If you install SQL Server 2019 (15.x) with the feature **Machine Learning Services and Language Extensions** or upgrade the SQL Server instance on a [Federal Information Processing Standard \(FIPS\)](#) enabled server, you will receive the following error:

*An error occurred while installing extensibility feature with error message:
AppContainer Creation Failed with error message NONE, state This implementation is not part of the Windows Platform FIPS validated cryptographic algorithms.*

Workaround

Disable FIPS before the installation of SQL Server 2019 (15.x) with the feature **Machine Learning Services and Language Extensions** or upgrade of the SQL Server instance. Once the installation or upgrade is complete, you can reenable FIPS.

Applies to: SQL Server 2019 (15.x)

R libraries using specific algorithms, streaming, or partitioning

Issue

The following limitations apply on SQL Server 2017 (14.x) with runtime upgrade. This issue applies to Enterprise edition.

- Parallelism: `RevoScaleR` and `MicrosoftML` algorithm thread parallelism for scenarios are limited to maximum of two threads.
- Streaming & partitioning: Scenarios involving `@r_rowsPerRead` parameter passed to T-SQL `sp_execute_external_script` isn't applied.
- Streaming & partitioning: `RevoScaleR` and `MicrosoftML` data sources (that is, `ODBC`, `XDF`) doesn't support reading rows in chunks for training or scoring scenarios. These scenarios always bring all data to memory for computation and the operations are memory bound

Solution

The best solution is to upgrade to SQL Server 2019 (15.x). Alternatively you can continue to use SQL Server 2017 (14.x) with runtime upgrade configured using [RegisterRext.exe /configure](#), after you complete the following tasks.

1. Edit registry to create a key

`Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\150` and

- add a value `SharedCode` with data `C:\Program Files\Microsoft SQL Server\150\Shared` or the instance shared directory, as configured.
2. Create a folder `C:\Program Files\Microsoft SQL Server\150\Shared` and copy `instapi140.dll` from the folder `C:\Program Files\Microsoft SQL Server\140\Shared` to the newly created folder.
 3. Rename the `instapi140.dll` to `instapi150.dll` in the new folder `C:\Program Files\Microsoft SQL Server\150\Shared`.

Important

If you do the steps above, you must manually remove the added key prior to upgrading to a later version of SQL Server.

Performance issues of Process Pooling in ML Services (R and Python)

This section contains known issues and workarounds for using ML services (R and Python) in SQL Server.

Cold start Performance of Process Pooling in ML Services

Upon execution of `sp_execute_external_script`, the launchpad service launches satellite processes that start the external runtimes such as R and Python. To amortize the startup cost, a pool of processes is created that can be used in the subsequent execution of `sp_execute_external_script`. This pool of processes is specific to this user, database, and the used language (R or Python in ML Services).

First query execution

The satellite processes need to be warmed up when `sp_execute_external_script` is executed for the first time or after a period of idle time (the processes are terminated via a cleanup task if they are not used for a while). Cold start of such pooled processes may be slow (for example, due to resource constraints).

Workaround

If the performance of the first call is important, it is recommended to keep the queries warm. For example, a background task can be executed that fires a simple

`sp_execute_external_script` query before the processes get expired. For instance, to keep R queries warm, you may execute the following query periodically.

SQL

```
EXECUTE sp_execute_external_script @language = N'R', @script = N'';
GO
```

High number of concurrent queries

If the number of concurrent execution of `sp_execute_external_script` is higher than the active R/Python processes in the pool, the cold start of adding additional processes to the pool may be slow (for example, due to resource constraints).

Workaround

To overcome the scaling performance issue, multiple requests can be batched (for example, via [loopback connections](#) or rewriting the script to handle multiple requests). In addition, for real-time scenarios [SQL PREDICT](#) can be utilized.

R script execution issues

This section contains known issues that are specific to running R on SQL Server, as well as some issues that are related to the R libraries and tools published by Microsoft, including RevoScaleR.

For additional known issues that might affect R solutions, see the [Machine Learning Server](#) site.

Access denied warning when executing R scripts on SQL Server in a non default location

If the instance of SQL Server has been installed to a non-default location, such as outside the `Program Files` folder, the warning `ACCESS_DENIED` is raised when you try to run scripts that install a package. For example:

```
In normalizePath(path.expand(path), winslash, mustWork) :  
path[2]="~\Externallibraries\R\8\1": Access is denied
```


The reason is that an R function attempts to read the path, and fails if the built-in users group `SQLRUserGroup`, doesn't have read access. The warning that is raised doesn't block execution of the current R script, but the warning might recur repeatedly whenever the user runs any other R script.

If you have installed SQL Server to the default location, this error doesn't occur, because all Windows users have read permissions on the `Program Files` folder.

This issue is addressed in an upcoming service release. As a workaround, provide the group, `SQLRUserGroup`, with read access for all parent folders of `ExternalLibraries`.

Serialization error between old and new versions of RevoScaleR

When you pass a model using a serialized format to a remote SQL Server instance, you might get the error:

```
Error in memDecompress(data, type = decompress) internal error -3 in  
memDecompress(2).
```

This error is raised if you saved the model using a recent version of the serialization function, `rxSerializeModel`, but the SQL Server instance where you deserialize the model has an older version of the RevoScaleR APIs, from SQL Server 2017 (14.x) CU 2 or earlier.

As a workaround, you can upgrade the SQL Server 2017 (14.x) instance to CU 3 or later.

The error doesn't appear if the API version is the same, or if you are moving a model saved with an older serialization function to a server that uses a newer version of the serialization API.

In other words, use the same version of RevoScaleR for both serialization and deserialization operations.

Real-time scoring doesn't correctly handle the *learningRate* parameter in tree and forest models

If you create a model using a decision tree or decision forest method and specify the learning rate, you might see inconsistent results when using `sp_rxpredict` or the SQL `PREDICT` function, as compared to using `rxPredict`.

The cause is an error in the API that processes serialized models, and is limited to the `learningRate` parameter: for example, in `rxBTrees`, or

This issue is addressed in an upcoming service release.

Limitations on processor affinity for R jobs

In the initial release build of SQL Server 2016 (13.x), you could set processor affinity only for CPUs in the first k-group. For example, if the server is a 2-socket machine with two k-groups, only processors from the first k-group are used for the R processes. The same limitation applies when you configure resource governance for R script jobs.

This issue is fixed in SQL Server 2016 (13.x) Service Pack 1. We recommend that you upgrade to the latest service release.

Applies to: SQL Server 2016 (13.x) R Services (RTM version)

Changes to column types can't be performed when reading data in a SQL Server compute context

If your compute context is set to the SQL Server instance, you can't use the *colClasses* argument (or other similar arguments) to change the data type of columns in your R code.

For example, the following statement would result in an error if the column `CRSDepTimeStr` isn't already an integer:

R

```
data <- RxSqlServerData(  
  sqlQuery = "SELECT CRSDepTimeStr, ArrDelay FROM AirlineDemoSmall",  
  connectionString = connectionString,  
  colClasses = c(CRSDepTimeStr = "integer"))
```

As a workaround, you can rewrite the SQL query to use `CAST` or `CONVERT` and present the data to R by using the correct data type. In general, performance is better when you work with data by using SQL rather than by changing data in the R code.

Applies to: SQL Server 2016 (13.x) R Services

Limits on size of serialized models

When you save a model to a SQL Server table, you must serialize the model and save it in a binary format. Theoretically the maximum size of a model that can be stored with this method is 2 GB, which is the maximum size of varbinary columns in SQL Server.

If you need to use larger models, the following workarounds are available:

- Take steps to reduce the size of your model. Some open source R packages include a great deal of information in the model object, and much of this information can be removed for deployment.
- Use feature selection to remove unnecessary columns.
- If you are using an open source algorithm, consider a similar implementation using the corresponding algorithm in MicrosoftML or RevoScaleR. These packages have been optimized for deployment scenarios.
- After the model has been rationalized and the size reduced using the preceding steps, see if the [memCompress](#) function in base R can be used to reduce the size of the model before passing it to SQL Server. This option is best when the model is close to the 2-GB limit.
- For larger models, you can use the SQL Server [FileTable](#) feature to store the models, rather than using a varbinary column.

To use FileTables, you must add a firewall exception, because data stored in FileTables is managed by the Filestream filesystem driver in SQL Server, and default firewall rules block network file access. For more information, see [Enable Prerequisites for FileTable](#).

After you have enabled FileTable, to write the model, you get a path from SQL using the FileTable API, and then write the model to that location from your code. When you need to read the model, you get the path from SQL Server, and then call the model using the path from your script. For more information, see [Access FileTables with File Input-Output APIs](#).

Avoid clearing workspaces when you execute R code in a SQL Server compute context

If you use an R command to clear your workspace of objects while running R code in a SQL Server compute context, or if you clear the workspace as part of an R script called by using [sp_execute_external_script](#), you might get this error: *workspace object revoScriptConnection not found*

`revoScriptConnection` is an object in the R workspace that contains information about an R session that is called from SQL Server. However, if your R code includes a command to clear the workspace (such as `rm(list=ls())`), all information about the session and other objects in the R workspace is cleared as well.

As a workaround, avoid indiscriminate clearing of variables and other objects while you're running R in SQL Server. Although clearing the workspace is common when working in the R console, it can have unintended consequences.

- To delete specific variables, use the R `remove` function: for example,

```
remove('name1', 'name2', ...)
```
- If there are multiple variables to delete, save the names of temporary variables to a list and perform periodic garbage collection.

Restrictions on data that can be provided as input to an R script

You can't use in an R script the following types of query results:

- Data from a Transact-SQL query that references AlwaysEncrypted columns.
- Data from a Transact-SQL query that references masked columns.

If you need to use masked data in an R script, a possible workaround is to make a copy of the data in a temporary table and use that data instead.

Use of strings as factors can lead to performance degradation

Using string type variables as factors can greatly increase the amount of memory used for R operations. This is a known issue with R in general, and there are many articles on the subject. For example, see [Factors aren't first-class citizens in R, by John Mount, in R-bloggers](#) or [stringsAsFactors: An unauthorized biography](#), by Roger Peng.

Although the issue isn't specific to SQL Server, it can greatly affect performance of R code run in SQL Server. Strings are typically stored as `varchar` or `nvarchar`, and if a column of string data has many unique values, the process of internally converting these to integers and back to strings by R can even lead to memory allocation errors.

If you don't absolutely require a string data type for other operations, mapping the string values to a numeric (integer) data type as part of data preparation would be beneficial from a performance and scale perspective.

For a discussion of this issue, and other tips, see [Performance for R Services - data optimization](#).

Arguments *varsToKeep* and *varsToDrop* aren't supported for SQL Server data sources

When you use the `rxDataStep` function to write results to a table, using the *varsToKeep* and *varsToDrop* is a handy way of specifying the columns to include or exclude as part of the operation. However, these arguments aren't supported for SQL Server data sources.

Limited support for SQL data types in `sp_execute_external_script`

Not all data types that are supported in SQL can be used in R. As a workaround, consider casting the unsupported data type to a supported data type before passing the data to `sp_execute_external_script`.

For more information, see [R libraries and data types](#).

Possible string corruption using Unicode strings in `varchar` columns

Passing Unicode data in `varchar` columns from SQL Server to R/Python can result in string corruption. This is due to the encoding for these Unicode strings in SQL Server collations may not match with the default UTF-8 encoding used in R/Python.

To send any non-ASCII string data from SQL Server to R/Python, use UTF-8 encoding (available in SQL Server 2019 (15.x)) or use `nvarchar` type for the same.

Only one value of type `raw` can be returned from `sp_execute_external_script`

When a binary data type (the R `raw` data type) is returned from R, the value must be sent in the output data frame.

With data types other than `raw`, you can return parameter values along with the results of the stored procedure by adding the `OUTPUT` keyword. For more information, see [Parameters](#).

If you want to use multiple output sets that include values of type `raw`, one possible workaround is to do multiple calls of the stored procedure, or to send the result sets back to SQL Server by using ODBC.

Loss of precision

Because Transact-SQL and R support various data types, numeric data types can suffer loss of precision during conversion.

For more information about implicit data-type conversion, see [R libraries and data types](#).

Variable scoping error when you use the transformFunc parameter

To transform data while you are modeling, you can pass a *transformFunc* argument in a function such as `rxLinmod` or `rxLogit`. However, nested function calls can lead to scoping errors in the SQL Server compute context, even if the calls work correctly in the local compute context.

The sample data set for the analysis has no variables

For example, assume that you have defined two functions, `f` and `g`, in your local global environment, and `g` calls `f`. In distributed or remote calls involving `g`, the call to `g` might fail with this error, because `f` can't be found, even if you have passed both `f` and `g` to the remote call.

If you encounter this problem, you can work around the issue by embedding the definition of `f` inside your definition of `g`, anywhere before `g` would ordinarily call `f`.

For example:

```
R

f <- function(x) { 2*x * 3 }
g <- function(y) {
  a <- 10 * y
  f(a)
}
```

To avoid the error, rewrite the definition as follows:

```
R

g <- function(y){
  f <- function(x) { 2*x +3}
  a <- 10 * y
  f(a)
}
```

Data import and manipulation using RevoScaleR

When `varchar` columns are read from a database, white space is trimmed. To prevent this, enclose strings in non-white-space characters.

When functions such as `rxDataStep` are used to create database tables that have `varchar` columns, the column width is estimated based on a sample of the data. If the width can vary, it might be necessary to pad all strings to a common length.

Using a transform to change a variable's data type isn't supported when repeated calls to `rxImport` or `rxTextToXdf` are used to import and append rows, combining multiple input files into a single `.xdf` file.

Limited support for `rxExec`

In SQL Server 2016 (13.x), the `rxExec` function that's provided by the RevoScaleR package can be used only in single-threaded mode.

Increase the maximum parameter size to support `rxGetVarInfo`

If you use data sets with extremely large numbers of variables (for example, over 40,000), set the `max-ppsize` flag when you start R to use functions such as `rxGetVarInfo`. The `max-ppsize` flag specifies the maximum size of the pointer protection stack.

If you are using the R console (for example, `RGui.exe` or `RTerm.exe`), you can set the value of `max-ppsize` to 500000 by typing:

```
R
```

```
R --max-ppsize=500000
```

Issues with the `rxDTree` function

The `rxDTree` function doesn't currently support in-formula transformations. In particular, using the `F()` syntax for creating factors on the fly isn't supported. However, numeric data is automatically binned.

Ordered factors are treated the same as factors in all RevoScaleR analysis functions except `rxDTree`.

data.table as an OutputDataSet in R

Using `data.table` as an `OutputDataSet` in R isn't supported in SQL Server 2017 (14.x) Cumulative Update 13 (CU 13) and earlier. The following message might appear:

text

```
Msg 39004, Level 16, State 20, Line 2
A 'R' script error occurred during execution of
'sp_execute_external_script' with HRESULT 0x80004004.
Msg 39019, Level 16, State 2, Line 2
An external script error occurred:
Error in alloc.col(newx) :
  Internal error: length of names (0) is not length of dt (11)
Calls: data.frame ... as.data.frame -> as.data.frame.data.table -> copy ->
alloc.col

Error in execution. Check the output for more information.
Error in eval(expr, envir, enclos) :
  Error in execution. Check the output for more information.
Calls: source -> withVisible -> eval -> eval -> .Call
Execution halted
```

`data.table` as an `OutputDataSet` in R is supported in SQL Server 2017 (14.x) Cumulative Update 14 (CU 14) and later.

Running a long script fails while installing a library

Running a long running external script session and having the dbo in parallel trying to install a library on a different database can terminate the script.

For example, running this external script against master:

SQL

```
USE MASTER
DECLARE @language nvarchar(1) = N'R'
DECLARE @script nvarchar(max) = N'Sys.sleep(100)'
DECLARE @input_data_1 nvarchar(max) = N'select 1'
EXEC sp_execute_external_script @language = @language, @script = @script, @input_data_1 = @input_data_1 with result sets none
go
```

While the dbo in parallel installs a library in LibraryManagementFunctional:

SQL


```
USE [LibraryManagementFunctional]
go

CREATE EXTERNAL LIBRARY [RODBC] FROM (CONTENT = N'/home/ani/var/opt/mssql/data/RODBC_1.3-16.tar.gz') WITH (LANGUAGE = 'R')
go

DECLARE @language nvarchar(1) = N'R'
DECLARE @script nvarchar(14) = N'library(RODBC)'
DECLARE @input_data_1 nvarchar(8) = N'select 1'
EXEC sp_execute_external_script @language = @language, @script = @script, @input_data_1 = @input_data_1
go
```

The previous long running external script against master will terminate with the following error message:

```
A 'R' script error occurred during execution of 'sp_execute_external_script' with
HRESULT 0x800704d4.
```

Workaround

Don't run the library install in parallel to the long-running query. Or rerun the long running query after the installation is complete.

Applies to: SQL Server 2019 (15.x) on Linux & Big Data Clusters only.

SQL Server stops responding when executing R scripts containing parallel execution

SQL Server 2019 (15.x) contains a regression that affects R scripts that use parallel execution. Examples include using `rxExec` with `RxLocalPar` compute context and scripts that use the parallel package. This problem is caused by errors the parallel package encounters when writing to the null device while executing in SQL Server.

Applies to: SQL Server 2019 (15.x).

Precision loss for money/numeric/decimal/bigint data types

Executing an R script with `sp_execute_external_script` allows money, numeric, decimal, and bigint data types as input data. However, because they are converted to R's numeric

type, they suffer a precision loss with values that are very high or have decimal point values.

- **money**: Sometimes cent values would be imprecise and a warning would be issued: *Warning: unable to precisely represent cents values.*
- **numeric/decimal**: `sp_execute_external_script` with an R script doesn't support the full range of those data types and would alter the last few decimal digits especially those with fraction.
- **bigint**: R only support up to 53-bit integers and then it will start to have precision loss.

Issues with the rxExecBy function - rxExecBy function cannot find installed package

When the `rxExecBy` function is called, a new R runtime process starts. This new process does not have updated library paths, hence, packages installed in locations other than the default library path are not found during execution.

Workaround

The path to R packages needs to be explicitly updated. Suppose the packages are installed in the external libraries path, the following R script could be used to update library path: `.libPaths(c(Sys.getenv("MRS_EXTLIB_USER_PATH"), Sys.getenv("MRS_EXTLIB_SHARED_PATH"), .libPaths()))`

Python script execution issues

This section contains known issues that are specific to running Python on SQL Server, as well as issues that are related to the Python packages published by Microsoft, including [revoscalepy](#) and [microsoftml](#).

Call to pretrained model fails if path to model is too long

If you installed the pretrained models in an early release of SQL Server 2017 (14.x), the complete path to the trained model file might be too long for Python to read. This limitation is fixed in a later service release.

There are several potential workarounds:

- When you install the pretrained models, choose a custom location.

- If possible, install the SQL Server instance under a custom installation path with a shorter path, such as `C:\SQL\MSSQL14.MSSQLSERVER`.
- Use the Windows utility `Fsutil` to create a hard link that maps the model file to a shorter path.
- Update to the latest service release.

Error when saving serialized model to SQL Server

When you pass a model to a remote SQL Server instance, and try to read the binary model using the `rx_unserialize` function in `revoscalepy`, you might get the error:

```
NameError: name 'rx_unserialize_model' is not defined
```

This error is raised if you saved the model using a recent version of the serialization function, but the SQL Server instance where you deserialize the model doesn't recognize the serialization API.

To resolve the issue, upgrade the SQL Server 2017 (14.x) instance to CU 3 or later.

Failure to initialize a varbinary variable causes an error in BxlServer

If you run Python code in SQL Server using `sp_execute_external_script`, and the code has output variables of type `varbinary(max)`, `varchar(max)` or similar types, the variable must be initialized or set as part of your script. Otherwise, the data exchange component, `BxlServer`, raises an error and stops working.

This limitation will be fixed in an upcoming service release. As a workaround, make sure that the variable is initialized within the Python script. Any valid value can be used, as in the following examples:

SQL

```
declare @b varbinary(max);
exec sp_execute_external_script
  @language = N'Python'
  , @script = N'b = 0x0'
  , @params = N'@b varbinary(max) OUTPUT'
  , @b = @b OUTPUT;
go
```

SQL

```
declare @b varchar(30);
exec sp_execute_external_script
    @language = N'Python'
    , @script = N' b = "" '
    , @params = N'@b varchar(30) OUTPUT'
    , @b = @b OUTPUT;
go
```

Telemetry warning on successful execution of Python code

Beginning with SQL Server 2017 (14.x) CU 2, the following message might appear even if Python code otherwise runs successfully:

STDERR message(s) from external script: ~PYTHON_SERVICES\lib\site-packages\revoscalepy\utils\RxTelemetryLogger SyntaxWarning: telemetry_state is used prior to global declaration

This issue has been fixed in SQL Server 2017 (14.x) Cumulative Update 3 (CU 3).

Numeric, decimal, and money data types not supported

Beginning with SQL Server 2017 (14.x) Cumulative Update 12 (CU 12), numeric, decimal and money data types in WITH RESULT SETS are unsupported when using Python with `sp_execute_external_script`. The following messages might appear:

[Code: 39004, SQL State: S1000] A 'Python' script error occurred during execution of 'sp_execute_external_script' with HRESULT 0x80004004.

[Code: 39019, SQL State: S1000] An external script error occurred:

SqlSatelliteCall error: Unsupported type in output schema. Supported types: bit, smallint, int, datetime, smallmoney, real and float. char, varchar are partially supported.

This has been fixed in SQL Server 2017 (14.x) Cumulative Update 14 (CU 14).

Bad interpreter error when installing Python packages with pip on Linux

On SQL Server 2019 (15.x), if you try to use `pip`. For example:

```
Bash
```

```
/opt/mssql/mlservices/runtime/python/bin/pip -h
```

You will then get this error:

```
bash: /opt/mssql/mlservices/runtime/python/bin/pip:  
/opt/microsoft/mlserver/9.4.7/bin/python/python: bad interpreter: No such file or  
directory
```

Workaround

Install **pip** from the [Python Package Authority \(PyPA\)](#) [↗]:

```
Bash
```

```
wget 'https://bootstrap.pypa.io/get-pip.py'  
/opt/mssql/mlservices/bin/python/python ./get-pip.py
```

Recommendation

See [Install Python packages with sqlmlutils](#).

Applies to: SQL Server 2019 (15.x) on Linux

Unable to install Python packages using **pip** after installing SQL Server 2019 on Windows

After installing SQL Server 2019 (15.x) on Windows, attempting to install a python package via **pip** from a DOS command line will fail. For example:

```
Bash
```

```
pip install quantfolio
```

This will return the following error:

```
pip is configured with locations that require TLS/SSL, however the ssl module in  
Python is not available.
```

This is a problem specific to the Anaconda package. It will be fixed in an upcoming service release.

Workaround

Copy the following files:

- `libssl-1_1-x64.dll`
- `libcrypto-1_1-x64.dll`

from the folder

```
C:\Program Files\Microsoft SQL
```

```
Server\MSSQL15.MSSQLSERVER\PYTHON_SERVICES\Library\bin
```

to the folder

```
C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\PYTHON_SERVICES\DLLs
```

Then open a new DOS command shell prompt.

Applies to: SQL Server 2019 (15.x) on Windows

Error when using `sp_execute_external_script` without `libc++abi.so` on Linux

On a clean Linux machine that doesn't have `libc++abi.so` installed, running a `sp_execute_external_script` (SPEES) query fails with a "No such file or directory" error.

For example:

SQL

```
EXEC sp_execute_external_script
    @language = N'Python'
    , @script = N'
OutputDataSet = InputDataSet'
    , @input_data_1 = N'select 1'
    , @input_data_1_name = N'InputDataSet'
    , @output_data_1_name = N'OutputDataSet'
    WITH RESULT SETS ([[output] int not null]);
```

Output

```
Msg 39012, Level 16, State 14, Line 0
Unable to communicate with the runtime for 'Python' script for request id:
94257840-1704-45E8-83D2-2F74AEB46CF7. Please check the requirements of
```

```
'Python' runtime.
STDERR message(s) from external script:

Failed to load library /opt/mssql-extensibility/lib/sqlsatellite.so with
error libc++abi.so.1: cannot open shared object file: No such file or
directory.

SqlSatelliteCall error: Failed to load library /opt/mssql-
extensibility/lib/sqlsatellite.so with error libc++abi.so.1: cannot open
shared object file: No such file or directory.
STDOUT message(s) from external script:
SqlSatelliteCall function failed. Please see the console output for more
information.
Traceback (most recent call last):
  File
"/opt/mssql/mlservices/libraries/PythonServer/revoscalepy/computecontext/RxI
nSqlServer.py", line 605, in rx_sql_satellite_call
    rx_native_call("SqlSatelliteCall", params)
  File
"/opt/mssql/mlservices/libraries/PythonServer/revoscalepy/RxSerializable.py"
, line 375, in rx_native_call
    ret = px_call(functionname, params)
RuntimeError: revoscalepy function failed.
Total execution time: 00:01:00.387
```

Workaround

Run the following command:

```
Bash
```

```
sudo cp /opt/mssql/lib/libc++abi.so.1 /opt/mssql-extensibility/lib/
```

Applies to: SQL Server 2019 (15.x) on Linux

Firewall rule creation error in `modprobe` when running `mssql-launchpadd` on Linux

When viewing the logs of `mssql-launchpadd` using `sudo journalctl -a -u mssql-launchpadd`, you might see a firewall rule creation error similar to the following output.

```
Output
```

```
-- Logs begin at Sun 2021-03-28 12:03:30 PDT, end at Wed 2022-10-12 13:20:17
PDT. --
Mar 22 16:57:51 sqlVm systemd[1]: Started Microsoft SQL Server Extensibility
Launchpad Daemon.
Mar 22 16:57:51 sqlVm launchpadd[195658]: 2022/03/22 16:57:51 [launchpadd]
```

```
INFO: Extensibility Log Header: <timestamp> <process> <sandboxId>
<sessionId> <message>
Mar 22 16:57:51 sqlVm launchpadd[195658]: 2022/03/22 16:57:51 [launchpadd]
INFO: No extensibility section in /var/opt/mssql/mssql.conf file. Using
default settings.
Mar 22 16:57:51 sqlVm launchpadd[195658]: 2022/03/22 16:57:51 [launchpadd]
INFO: DataDirectories =
/bin:/etc:/lib:/lib32:/lib64:/sbin:/usr/bin:/usr/include:/usr/lib:/usr/lib32
:/usr/lib64:/usr/libexec/gcc:/usr/sbin:/usr/share:/var/lib:/opt/microsoft:/o
pt/mssql-extensibility:/opt/mssql/mlservices:/opt/mssql/lib/zulu-jre-
11:/opt/mssql-tools
Mar 22 16:57:51 sqlVm launchpadd[195658]: 2022/03/22 16:57:51 [launchpadd]
INFO: [RG] SQL Extensibility Cgroup initialization is done.
Mar 22 16:57:51 sqlVm launchpadd[195658]: 2022/03/22 16:57:51 [launchpadd]
INFO: Found 1 IP address(es) from the bridge.
Mar 22 16:57:51 sqlVm launchpadd[195676]: modprobe: ERROR: could not insert
'ip6_tables': Operation not permitted
Mar 22 16:57:51 sqlVm launchpadd[195673]: ip6tables v1.8.4 (legacy): can't
initialize ip6tables table `filter': Table does not exist (do you need to
insmod?)
Mar 22 16:57:51 sqlVm launchpadd[195673]: Perhaps ip6tables or your kernel
needs to be upgraded.
Mar 22 16:57:51 sqlVm launchpadd[195678]: modprobe: ERROR: could not insert
'ip6_tables': Operation not permitted
Mar 22 16:57:51 sqlVm launchpadd[195677]: ip6tables v1.8.4 (legacy): can't
initialize ip6tables table `filter': Table does not exist (do you need to
insmod?)
Mar 22 16:57:51 sqlVm launchpadd[195677]: Perhaps ip6tables or your kernel
needs to be upgraded.
Mar 22 16:57:51 sqlVm launchpadd[195670]: 2022/03/22 16:57:51 [setnetbr]
ERROR: Failed to set firewall rules: exit status 3
```

Workaround

Run the following commands to configure `modprobe`, and restart the SQL Server Launchpad service:

```
Bash
```

```
sudo modprobe ip6_tables
sudo systemctl restart mssql-launchpadd
```

Applies to: SQL Server 2019 (15.x) and later on Linux

Can't install `tensorflow` package using `sqlmlutils`

The `sqlmlutils` package is used to install Python packages in SQL Server 2019 (15.x). You need to download, install, and update the [Microsoft Visual C++ 2015-2019](#)

[Redistributable \(x64\)](#). However, the `tensorflow` package can't be installed using `sqlmlutils`. The `tensorflow` package depends on a newer version of `numpy` than the version installed in SQL Server. However, `numpy` is a preinstalled system package that `sqlmlutils` can't update when trying to install `tensorflow`.

Workaround

Using a command prompt in administrator mode, run the following command, replacing "MSSQLSERVER" with the name of your SQL instance:

Windows Command Prompt

```
"C:\Program Files\Microsoft SQL  
Server\MSSQL15.MSSQLSERVER\PYTHON_SERVICES\python.exe" -m pip install --  
upgrade tensorflow
```

If you get a "TLS/SSL" error, see [7. Unable to install Python packages using pip](#) earlier in this article.

Applies to: SQL Server 2019 (15.x) on Windows

Revolution R Enterprise and Microsoft R Open

This section lists issues specific to R connectivity, development, and performance tools that are provided by Revolution Analytics. These tools were provided in earlier pre-release versions of SQL Server.

In general, we recommend that you uninstall these previous versions and install the latest version of SQL Server or Microsoft R Server.

Revolution R Enterprise isn't supported

Installing Revolution R Enterprise side by side with any version of R Services (In-Database) isn't supported.

If you have an existing license for Revolution R Enterprise, you must put it on a separate computer from both the SQL Server instance and any workstation that you want to use to connect to the SQL Server instance.

Some pre-release versions of R Services (In-Database) included an R development environment for Windows that was created by Revolution Analytics. This tool is no longer provided, and isn't supported.

For compatibility with R Services (In-Database), we recommend that you install Microsoft R Client instead. [R Tools for Visual Studio](#) and [Visual Studio Code](#) also supports Microsoft R solutions.

Compatibility issues with SQLite ODBC driver and RevoScaleR

Revision 0.92 of the SQLite ODBC driver is incompatible with RevoScaleR. Revisions 0.88-0.91 and 0.93 and later are known to be compatible.

Next steps

- [Collect data to troubleshoot SQL Server Machine Learning Services](#)

Collect data to troubleshoot Python and R scripts with SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

Important


The support for Machine Learning Server (previously known as R Server) ended on July 1, 2022. For more information, see [What's happening to Machine Learning Server?](#)

This article describes how to collect the data you need when you're attempting to resolve problems in [SQL Server Machine Learning Services](#). This data can be useful whether you're resolving problems on your own or with the help of Microsoft customer support.

SQL Server version and edition

SQL Server 2016 R Services is the first release of SQL Server to include integrated R support. SQL Server 2016 Service Pack 1 (SP1) includes several major improvements, including the ability to run external scripts. If you are using SQL Server 2016, you should consider installing SP1 or later.

SQL Server 2017 and later has Python language integration. You cannot get Python feature integration in earlier releases.

For assistance getting edition and versions, see this article, which lists the build numbers for each of the [SQL Server versions](#) .

Depending on the edition of SQL Server you're using, some machine learning functionality might be unavailable, or limited.

R language and tool versions

In general, the version of Microsoft R that is installed when you select the R Services feature or the Machine Learning Services feature is determined by the SQL Server build

number. If you upgrade or patch SQL Server, you must also upgrade or patch its R components.

For a list of releases and links to R component downloads, see [Install machine learning components without internet access](#). On computers with internet access, the required version of R is identified and installed automatically.

It's possible to upgrade the R Server components separately from the SQL Server database engine, in a process known as binding. Therefore, the version of R that you use when you run R code in SQL Server might differ depending on both the installed version of SQL Server and whether you have migrated the server to the latest R version.

Determine the R version

The easiest way to determine the R version is to get the runtime properties by running a statement such as the following:

SQL

```
EXECUTE sp_execute_external_script
    @language = N'R'
    , @script = N'
# Transform R version properties to data.frame
OutputDataSet <- data.frame(
    property_name = c("R.version", "Revo.version"),
    property_value = c(R.Version()$version.string,
Revo.version$version.string),
    stringsAsFactors = FALSE)
# Retrieve properties like R.home, libPath & default packages
OutputDataSet <- rbind(OutputDataSet, data.frame(
    property_name = c("R.home", "libPaths", "defaultPackages"),
    property_value = c(R.home(), .libPaths(),
paste(getOption("defaultPackages"), collapse=", ")),
    stringsAsFactors = FALSE)
)
'
WITH RESULT SETS ((PropertyName nvarchar(100), PropertyValue
nvarchar(4000)));
```

Tip

If R Services is not working, try running only the R script portion from RGui.

As a last resort, you can open files on the server to determine the installed version. To do so, locate the `launcher.config` file to get the location of the R runtime and the

current working directory. We recommend that you make and open a copy of the file so that you don't accidentally change any properties.

- SQL Server 2016

```
C:\Program Files\Microsoft SQL Server\MSSQL13.  
<instance_name\MSSQL\Binn\rlauncher.config
```

- SQL Server 2017

```
C:\Program Files\Microsoft SQL Server\MSSQL14.  
<instance_name>\MSSQL\Binn\rlauncher.config
```

To get the R version and RevoScaleR versions, open an R command prompt, or open the RGui that's associated with the instance.

- SQL Server 2016

```
C:\Program Files\Microsoft SQL Server\MSSQL13.  
<instancename>\R_SERVICES\bin\x64\RGui.exe
```

- SQL Server 2017

```
C:\Program Files\Microsoft SQL Server\MSSQL14.  
<instance_name>\R_SERVICES\bin\x64\RGui.exe
```

The R console displays the version information on startup. For example, the following version represents the default configuration for SQL Server 2017:

```
Console  
  
*Microsoft R Open 3.3.3*  
  
*The enhanced R distribution from Microsoft*  
  
*Microsoft packages Copyright (C) 2017 Microsoft*  
  
*Loading Microsoft R Server packages, version 9.1.0.*
```

Python versions

There are several ways to get the Python version. The easiest way is to run this statement from Management Studio or any other SQL query tool:

```
SQL
```

```

-- Get Python runtime properties:
exec sp_execute_external_script
    @language = N'Python'
    , @script = N'
import sys
import pkg_resources
OutputDataSet = pandas.DataFrame(
    {"property_name": ["Python.home", "Python.version",
"Revo.version", "libpaths"],
    "property_value": [sys.executable[:-10], sys.version,
pkg_resources.get_distribution("revoscalepy").version, str(sys.path)]}
)
'
with WITH RESULT SETS (SQL keywords) ((PropertyName nvarchar(100),
PropertyValue nvarchar(4000)));

```

If Machine Learning Services is not running, you can determine the installed Python version by looking at the `pythonlauncher.config` file. We recommend that you make and open a copy of the file so that you don't accidentally change any properties.

1. For SQL Server 2017 only: `C:\Program Files\Microsoft SQL Server\MSSQL14.<instance_name>\MSSQL\Log\ExtensibilityLog\pythonlauncher.config`
2. Get the value for `PYTHONHOME`.
3. Get the value of the current working directory.

ⓘ Note

If you have installed both Python and R in SQL Server 2017, the working directory and the pool of worker accounts are shared for the R and Python languages.

Are multiple instances of R or Python installed?

Check to see whether more than one copy of the R libraries is installed on the computer. This duplication can happen if:

- During setup you select both R Services (In-Database) and R Server (Standalone).
- You install Microsoft R Client in addition to SQL Server.
- A different set of R libraries was installed by using R Tools for Visual Studio, R Studio, Microsoft R Client, or another R IDE.
- The computer hosts multiple instances of SQL Server, and more than one instance uses machine learning.

The same conditions apply to Python.

If you find that multiple libraries or runtimes are installed, make sure that you get only the errors associated with the Python or R runtimes that are used by the SQL Server instance.

Origin of errors

The errors that you see when you attempt to run R code can come from any of the following sources:

- SQL Server database engine, including the stored procedure `sp_execute_external_script`
- The SQL Server Trusted Launchpad
- Other components of the extensibility framework, including R and Python launchers and satellite processes
- Providers, such as Microsoft Open Database Connectivity (ODBC)
- R language

When you work with the service for the first time, it can be difficult to tell which messages originate from which services. We recommend that you capture not only the exact message text, but the context in which you saw the message. Note the client software that you're using to run machine learning code:

- Are you using Management Studio? An external application?
- Are you running R code in a remote client, or directly in a stored procedure?

SQL Server log files

Get the most recent SQL Server ERRORLOG. The complete set of error logs consists of the files from the following default log directory:

- SQL Server 2016

```
C:\Program Files\Microsoft SQL  
Server\MSSQL13.SQL2016\MSSQL\Log\ExtensibilityLog
```

- SQL Server 2017

```
C:\Program Files\Microsoft SQL  
Server\MSSQL14.SQL2016\MSSQL\Log\ExtensibilityLog
```

ⓘ **Note**

The exact folder name differs depending on the instance name.

Errors returned by sp_execute_external_script

Get the complete text of errors that are returned, if any, when you run the `sp_execute_external_script` command.

To remove R or Python problems from consideration, you can run this script, which starts the R or Python runtime and passes data back and forth.

For R

SQL

```
exec sp_execute_external_script @language =N'R',
@script=N'OutputDataSet<-InputDataSet',
@input_data_1 =N'select 1 as hello'
with result sets ([[hello] int not null));
go
```

For Python

SQL

```
exec sp_execute_external_script @language =N'Python',
@script=N'OutputDataSet= InputDataSet',
@input_data_1 =N'select 1 as hello'
with result sets ([[hello] int not null));
go
```

Errors generated by the extensibility framework

SQL Server generates separate logs for the external script language runtimes. These errors are not generated by the Python or R language. They're generated from the extensibility components in SQL Server, including language-specific launchers and their satellite processes.

You can get these logs from the following default locations:

- SQL Server 2016

```
C:\Program Files\Microsoft SQL Server\MSSQL13.
<instance_name>\MSSQL\Log\ExtensibilityLog
```


- SQL Server 2017

```
C:\Program Files\Microsoft SQL Server\MSSQL14.
```

```
<instance_name>\MSSQL\Log\ExtensibilityLog
```

ⓘ Note

The exact folder name differs based on the instance name. Depending on your configuration, the folder might be on a different drive.

For example, the following log messages are related to the extensibility framework:

- *LogonUser Failed for user MSSQLSERVER01*

This might indicate that the worker accounts that run external scripts cannot access the instance.

- *InitializePhysicalUsersPool Failed*

This message might mean that your security settings are preventing setup from creating the pool of worker accounts that are needed to run external scripts.

- *Security Context Manager initialization failed*
- *Satellite Session Manager initialization failed*

System events

1. Open Windows Event Viewer, and search the **System Event** log for messages that include the string *Launchpad*.
2. Open the ExtLaunchErrorlog file, and look for the string *ErrorCode*. Review the message that's associated with the ErrorCode.

For example, the following messages are common system errors that are related to the SQL Server extensibility framework:

- *The SQL Server Launchpad (MSSQLSERVER) service failed to start due to the following error: <text>*
- *The service did not respond to the start or control request in a timely fashion.*
- *A timeout was reached (120000 milliseconds) while waiting for the SQL Server Launchpad (MSSQLSERVER) service to connect.*

Dump files

If you are knowledgeable about debugging, you can use the dump files to analyze a failure in Launchpad.

1. Locate the folder that contains the setup bootstrap logs for SQL Server. For example, in SQL Server 2016, the default path was C:\Program Files\Microsoft SQL Server\130\Setup Bootstrap\Log.
2. Open the bootstrap log subfolder that is specific to extensibility.
3. If you need to submit a support request, add the entire contents of this folder to a zipped file. For example, C:\Program Files\Microsoft SQL Server\130\Setup Bootstrap\Log\LOG\ExtensibilityLog.

The exact location might differ on your system, and it might be on a drive other than your C drive. Be sure to get the logs for the instance where machine learning is installed.

Configuration settings

This section lists additional components or providers that can be a source of errors when you run R or Python scripts.

What network protocols are available?

Machine Learning Services requires the following network protocols for internal communication among extensibility components, and for communication with external R or Python clients.

- Named pipes
- TCP/IP

Open SQL Server Configuration Manager to determine whether a protocol is installed and, if it is installed, to determine whether it is enabled.

Security configuration and permissions

For worker accounts:

1. In Control Panel, open **Users and Groups**, and locate the group used to run external script jobs. By default, the group is **SQLRUserGroup**.
2. Verify that the group exists and that it contains at least one worker account.
3. In SQL Server Management Studio, select the instance where R or Python jobs will be run, select **Security**, and then determine whether there is a logon for

SQLRUserGroup.

4. Review permissions for the user group.

For individual user accounts:

1. Determine whether the instance supports Mixed Mode authentication, SQL logins only, or Windows authentication only. This setting affects your R or Python code requirements.
2. For each user who needs to run R code, determine the required level of permissions on each database where objects will be written from R, data will be accessed, or objects will be created.
3. To enable script execution, create roles or add users to the following roles, as necessary:
 - All but *db_owner*: Require EXECUTE ANY EXTERNAL SCRIPT.
 - *db_datawriter*: To write results from R or Python.
 - *db_ddladmin*: To create new objects.
 - *db_datareader*: To read data that's used by R or Python code.
4. Note whether you changed any default startup accounts when you installed SQL Server 2016.
5. If a user needs to install new R packages or use R packages that were installed by other users, you might need to enable package management on the instance and then assign additional permissions.

What folders are subject to locking by antivirus software?

Antivirus software can lock folders, which prevents both the setup of the machine learning features and successful script execution. Determine whether any folders in the SQL Server tree are subject to virus scanning.

However, when multiple services or features are installed on an instance, it can be difficult to enumerate all possible folders that are used by the instance. For example, when new features are added, the new folders must be identified and excluded.

Moreover, some features create new folders dynamically at runtime. For example, in-memory OLTP tables, stored procedures, and functions all create new directories at runtime. These folder names often contain GUIDs and cannot be predicted. The SQL Server Trusted Launchpad creates new working directories for R and Python script jobs.

Because it might not be possible to exclude all folders that are needed by the SQL Server process and its features, we recommend that you exclude the entire SQL Server instance directory tree.

Is the firewall open for SQL Server? Does the instance support remote connections?

1. To determine whether SQL Server supports remote connections, see [Configure remote server connections](#).
2. Determine whether a firewall rule has been created for SQL Server. For security reasons, in a default installation, it might not be possible for remote R or Python client to connect to the instance. For more information, see [Troubleshooting connecting to SQL Server](#).

See also

[Troubleshoot machine learning in SQL Server](#)

Troubleshoot issues with Launchpad service executing Python and R scripts in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This article provides troubleshooting guidance for issues involving the [SQL Server Launchpad service](#) used with [Machine Learning Services](#). The Launchpad service supports external script execution for R and Python. Multiple issues can prevent Launchpad from starting, including configuration problems or changes, or missing network protocols.

Determine whether Launchpad is running

1. Open [SQL Server Configuration Manager](#). From the command line, type `SQLServerManager13.msc`, `SQLServerManager14.msc`, or `SQLServerManager15.msc`.
2. Make a note of the service account that Launchpad is running under. Each instance where R or Python is enabled should have its own instance of the Launchpad service. For example, the service for a named instance might be something like `MSSQLLaunchpad$InstanceName`.
3. If the service is stopped, restart it. On restarting, if there are any issues with configuration, a message is published in the system event log, and the service is stopped again. Check the system event log for details about why the service stopped.
4. Review the contents of `RSetup.log`, and make sure that there are no errors in the setup. For example, the message *Exiting with code 0* indicates failure of the service to start.
5. To look for other errors, review the contents of `launcher.log`.

Check the Launchpad service account

The default service account might be "NT Service\$SQL2016", "NT Service\$SQL2017", or "NT Service\$SQL2019". The final part might vary, depending on your SQL instance name.

The Launchpad service (Launchpad.exe) runs by using a low-privilege service account. However, to start R and Python and communicate with the database instance, the Launchpad service account requires the following user rights:

- Log on as a service (SeServiceLogonRight)
- Replace a process-level token (SeAssignPrimaryTokenPrivilege)
- Bypass traverse checking (SeChangeNotifyPrivilege)
- Adjust memory quotas for a process (SeIncreaseQuotaSizePrivilege)

For information about these user rights, see the "Windows privileges and rights" section in [Configure Windows service accounts and permissions](#).

Tip

If you are familiar with the use of the Support Diagnostics Platform (SDP) tool for SQL Server diagnostics, you can use SDP to review the output file with the name `MachineName_UserRights.txt`.

User group for Launchpad cannot log on locally

During setup of Machine Learning Services, SQL Server creates the Windows user group **SQLRUserGroup** and then provisions it with all rights necessary for Launchpad to connect to SQL Server and run external script jobs. If this user group is enabled, it is also used to execute Python scripts.

However, in organizations where more restrictive security policies are enforced, the rights that are required by this group might have been manually removed, or they might be automatically revoked by policy. If the rights have been removed, Launchpad can no longer connect to SQL Server, and SQL Server cannot call the external runtime.

To correct the problem, ensure that the group **SQLRUserGroup** has the system right **Allow log on locally**.

For more information, see [Configure Windows service accounts and permissions](#).

Permissions to run external scripts

Even if Launchpad is configured correctly, it returns an error if the user does not have permission to run R or Python scripts.

If you installed SQL Server as a database administrator or you are a database owner, you are automatically granted this permission. However, other users usually have more limited permissions. If they try to run an R script, they get a Launchpad error.

To correct the problem, in SQL Server Management Studio, a security administrator can modify the SQL login or Windows user account by running the following script:

```
SQL
```

```
GRANT EXECUTE ANY EXTERNAL SCRIPT TO <username>
```

For more information, see [GRANT \(Transact-SQL\)](#).

Common Launchpad errors

This section lists the most common error messages that Launchpad returns.

"Unable to launch runtime for R script"

If the Windows group for R users (also used for Python) cannot log on to the instance that is running R Services, you might see the following errors:

- Errors generated when you try to run R scripts:
 - *Unable to launch runtime for 'R' script. Please check the configuration of the 'R' runtime.*
 - *An external script error occurred. Unable to launch the runtime.*
- Errors generated by the SQL Server Launchpad service:
 - *Failed to initialize the launcher RLauncher.dll*
 - *No launcher dlls were registered!*
 - *Security logs indicate that the account NT SERVICE was unable to log on*

For information about how to grant this user group the necessary permissions, see [Install SQL Server R Services](#).

ⓘ Note

This limitation does not apply if you use SQL logins to run R scripts from a remote workstation.

"Logon failure: the user has not been granted the requested logon type"

By default, SQL Server Launchpad uses the following account on startup: `NT Service\MSSQLLaunchpad`. The account is configured by SQL Server setup to have all necessary permissions.

If you assign a different account to Launchpad, or the right is removed by a policy on the SQL Server machine, the account might not have the necessary permissions, and you might see this error:

```
ERROR_LOGON_TYPE_NOT_GRANTED 1385 (0x569) Logon failure: the user has not been granted the requested logon type at this computer.
```

To grant the necessary permissions to the new service account, use the Local Security Policy application, and update the permissions on the account to include the following permissions:

- Adjust memory quotas for a process (SeIncreaseQuotaPrivilege)
- Bypass traverse checking (SeChangeNotifyPrivilege)
- Log on as a service (SeServiceLogonRight)
- Replace a process-level token (SeAssignPrimaryTokenPrivilege)

"Unable to communicate with the Launchpad service"

If you have installed and then enabled machine learning, but you get this error when you try to run an R or Python script, the Launchpad service for the instance might have stopped running.

1. From a Windows command prompt, open the SQL Server Configuration Manager. For more information, see [SQL Server Configuration Manager](#).
2. Right-click SQL Server Launchpad for the instance, and then select **Properties**.
3. Select the **Service** tab, and then verify that the service is running. If it is not running, change the **Start Mode** to **Automatic**, and then select **Apply**.

4. Restarting the service usually fixes the problem so that machine learning scripts can run. If the restart does not fix the issue, note the path and the arguments in the **Binary Path** property, and do the following:
 - a. Review the launcher's .config file and ensure that the working directory is valid.
 - b. Ensure that the Windows group that's used by Launchpad can connect to the SQL Server instance.
 - c. If you change any of the service properties, restart the Launchpad service.

"Fatal error creation of tmpFile failed"

In this scenario, you have successfully installed machine learning features, and Launchpad is running. You try to run some simple R or Python code, but Launchpad fails with an error like the following:

Unable to communicate with the runtime for R script. Please check the requirements of R runtime.

At the same time, the external script runtime writes the following message as part of the STDERR message:

Fatal error: creation of tmpfile failed.

This error indicates that the account that Launchpad is attempting to use does not have permission to log on to the database. This situation can happen when strict security policies are implemented. To determine whether this is the case, review the SQL Server logs, and check to see whether the MSSQLSERVER01 account was denied at login. The same information is provided in the logs that are specific to R_SERVICES or PYTHON_SERVICES. Look for ExtLaunchError.log.

By default, 20 accounts are set up and associated with the Launchpad.exe process, with the names MSSQLSERVER01 through MSSQLSERVER20. If you make heavy use of R or Python, you can increase the number of accounts.

To resolve the issue, ensure that the group has *Allow Log on Locally* permissions to the local instance where machine learning features have been installed and enabled. In some environments, this permission level might require a GPO exception from the network administrator.

"Not enough quota to process this command"

This error can mean one of several things:

- Launchpad might have insufficient external users to run the external query. For example, if you are running more than 20 external queries concurrently, and there are only 20 default users, one or more queries might fail.
- Insufficient memory is available to process the R task. This error happens most often in a default environment, where SQL Server might be using up to 70 percent of the computer's resources. For information about how to modify the server configuration to support greater use of resources by R, see [Operationalizing your R code](#).

"Can't find package"

If you run R code in SQL Server and get this message, but did not get the message when you ran the same code outside SQL Server, it means that the package was not installed to the default library location used by SQL Server.

This error can happen in many ways:

- You installed a new package on the server, but access was denied, so R installed the package to a user library.
- You installed R Services and then installed another R tool or set of libraries, such as RStudio.

To determine the location of the R package library that's used by the instance, open SQL Server Management Studio (or any other database query tool), connect to the instance, and then run the following stored procedure:

SQL

```
EXEC sp_execute_external_script @language = N'R',  
@script = N' print(normalizePath(R.home())); print(.libPaths());';
```

Sample results

STDOUT message(s) from external script:

```
[1] "C:\Program Files\Microsoft SQL Server\MSSQL13.SQL2016\R_SERVICES"
```

[1] "C:/Program Files/Microsoft SQL Server/MSSQL13.SQL2016/R_SERVICES/library"

To resolve the issue, you must reinstall the package to the SQL Server instance library.

ⓘ Note

If you have upgraded an instance of SQL Server 2016 to use the latest version of Microsoft R, the default library location is different. For more information, see [Default R library location](#).

Launchpad shuts down due to mismatched DLLs

If you install the database engine with other features, patch the server, and then later add the Machine Learning feature by using the original media, the wrong version of the Machine Learning components might be installed. When Launchpad detects a version mismatch, it shuts down and creates a dump file.

To avoid this problem, be sure to install any new features at the same patch level as the server instance.

The wrong way to upgrade:

1. Install SQL Server 2016 without R Services.
2. Upgrade SQL Server 2016 Cumulative Update 2.
3. Install R Services (In-Database) by using the RTM media.

The correct way to upgrade:

1. Install SQL Server 2016 without R Services.
2. Upgrade SQL Server 2016 to the desired patch level. For example, install Service Pack 1 and then Cumulative Update 2.
3. To add the feature at the correct patch level, run SP1 and CU2 setup again, and then choose R Services (In-Database).

Launchpad fails to start if 8dot3 notation is required

ⓘ Note

On older systems, Launchpad can fail to start if there is an 8dot3 notation requirement. This requirement has been removed in later releases. SQL Server 2016 R Services customers should install one of the following:

- SQL Server 2016 SP1 and CU1: [Cumulative Update 1 for SQL Server](#) .
- SQL Server 2016 RTM, Cumulative Update 3, and this [hotfix](#) , which is available on demand.

For compatibility with R, SQL Server 2016 R Services (In-Database) required the drive where the feature is installed to support the creation of short file names by using *8dot3 notation*. An 8.3 file name is also called a *short file name*, and it's used for compatibility with earlier versions of Microsoft Windows or as an alternative to long file names.

If the volume where you are installing R does not support short file names, the processes that launch R from SQL Server might not be able to locate the correct executable, and Launchpad will not start.

As a workaround, you can enable the 8dot3 notation on the volume where SQL Server is installed and where R Services is installed. You must then provide the short name for the working directory in the R Services configuration file.

1. To enable 8dot3 notation, run the fsutil utility with the *8dot3name* argument as described here: [fsutil 8dot3name](#).
2. After the 8dot3 notation is enabled, open the RLauncher.config file and note the property of `WORKING_DIRECTORY`. For information about how to find this file, see [Data collection for Machine Learning troubleshooting](#).
3. Use the fsutil utility with the *file* argument to specify a short file path for the folder that's specified in `WORKING_DIRECTORY`.
4. Edit the configuration file to specify the same working directory that you entered in the `WORKING_DIRECTORY` property. Alternatively, you can specify a different working directory and choose an existing path that's already compatible with the 8dot3 notation.

Next steps

[Data collection for troubleshooting machine learning](#)

[Install SQL Server Machine Learning Services](#)

[Troubleshoot database engine connections](#)

Common R script errors in SQL Server Machine Learning Services

Article • 03/03/2023

Applies to:  SQL Server 2016 (13.x) and later versions

This article documents several common script errors when running R script in [SQL Server Machine Learning Services](#). The list is not comprehensive. There are many packages and errors can vary between versions of the same package.

Valid script fails in T-SQL or in stored procedures

Before wrapping your R code in a stored procedure, it is a good idea to run your R code in an external IDE, or in one of the R tools such as RTerm or RGui. By using these methods, you can test and debug the code by using the detailed error messages that are returned by R.

However, sometimes code that works perfectly in an external IDE or utility might fail to run in a stored procedure or in a SQL Server compute context. If this happens, there are a variety of issues to look for before you can assume that the package doesn't work in SQL Server.

1. Check to see whether Launchpad is running.
2. Review messages to see whether either the input data or output data contains columns with incompatible or unsupported data types. For example, queries on a SQL database often return GUIDs or RowGUIDs, both of which are unsupported. For more information, see [R libraries and data types](#).
3. Review the help pages for individual R functions to determine whether all parameters are supported for the SQL Server compute context. For ScaleR help, use the inline R help commands, or see [Package Reference](#).

If the R runtime is functioning but your script returns errors, we recommend that you try debugging the script in a dedicated R development environment, such as R Tools for Visual Studio.

We also recommend that you review and slightly rewrite the script to correct any problems with data types that might arise when you move data between R and the database engine. For more information, see [R libraries and data types](#).

Additionally, you can use the `sqlrutils` package to bundle your R script in a format that is more easily consumed as a stored procedure. For more information, see:

- [sqlrutils package](#)
- [Create a stored procedure by using sqlrutils](#)

Script returns inconsistent results

R scripts can return different values in a SQL Server context, for several reasons:

- Implicit type conversion is automatically performed on some data types, when the data is passed between SQL Server and R. For more information, see [R libraries and data types](#).
- Determine whether bitness is a factor. For example, there are often differences in the results of math operations for 32-bit and 64-bit floating point libraries.
- Determine whether NaNs were produced in any operation. This can invalidate results.
- Small differences can be amplified when you take a reciprocal of a number near zero.
- Accumulated rounding errors can cause such things as values that are less than zero instead of zero.

Implied authentication for remote execution via ODBC

If you connect to the SQL Server computer to run R commands by using the `RevoScaleR` functions, you might get an error when you use ODBC calls that write data to the server. This error happens only when you're using Windows authentication.

The reason is that the worker accounts that are created for R Services do not have permission to connect to the server. Therefore, ODBC calls cannot be executed on your behalf. The problem does not occur with SQL logins because, with SQL logins, the credentials are passed explicitly from the R client to the SQL Server instance and then to ODBC. However, using SQL logins is also less secure than using Windows authentication.

To enable your Windows credentials to be passed securely from a script that's initiated remotely, SQL Server must emulate your credentials. This process is termed *implied*

authentication. To make this work, the worker accounts that run R or Python scripts on the SQL Server computer must have the correct permissions.

1. Open SQL Server Management Studio as an administrator on the instance where you want to run R code.
2. Run the following script. Be sure to edit the user group name, if you changed the default, and the computer and instance names.

SQL

```
USE [master]
GO
```

```
CREATE LOGIN [computername\SQLRUserGroup] FROM WINDOWS WITH
DEFAULT_DATABASE=[master], DEFAULT_LANGUAGE=[language]
GO
```

Avoid clearing the workspace while you're running R in a SQL compute context

Although clearing the workspace is common when you work in the R console, it can have unintended consequences in a SQL compute context.

`revoScriptConnection` is an object in the R workspace that contains information about an R session that's called from SQL Server. However, if your R code includes a command to clear the workspace (such as `rm(list=ls())`), all information about the session and other objects in the R workspace is cleared as well.

As a workaround, avoid indiscriminate clearing of variables and other objects while you're running R in SQL Server. You can delete specific variables by using the **remove** function:

R

```
remove('name1', 'name2', ...)
```

If there are multiple variables to delete, we suggest that you save the names of temporary variables to a list and then perform periodic garbage collections on the list.

Next steps

Data collection for troubleshooting SQL Server Machine Learning Services

Install SQL Server Machine Learning Services

Troubleshoot database engine connections