

Contents

[Machine Learning Studio \(classic\) Module Reference](#)

[A-Z Module List](#)

[Module Categories and Descriptions](#)

[Data Format Conversions](#)

[Convert to ARFF](#)

[Convert to CSV](#)

[Convert to Dataset](#)

[Convert to SVMLight](#)

[Convert to TSV](#)

[Data Input and Output](#)

[Enter Data Manually](#)

[Export Data](#)

[Export to Hive Query](#)

[Export to Azure SQL Database](#)

[Export to Azure Table](#)

[Export to Azure Blob Storage](#)

[Import Data](#)

[Import from Web URL via HTTP](#)

[Import from Hive Query](#)

[Import from Azure SQL Database](#)

[Import from Azure Table](#)

[Import from Azure Blob Storage](#)

[Import from Data Feed Providers](#)

[Import from On-Premises SQL Server Database](#)

[Import from Azure Cosmos DB](#)

[Load Trained Model](#)

[Unpack Zipped Datasets](#)

[Data Transformation](#)

[Filter](#)

- [Apply Filter](#)
- [FIR Filter](#)
- [IIR Filter](#)
- [Median Filter](#)
- [Moving Average Filter](#)
- [Threshold Filter](#)
- [User-Defined Filter](#)
- [Learning with Counts](#)
 - [Build Counting Transform](#)
 - [Export Count Table](#)
 - [Import Count Table](#)
 - [Merge Count Transform](#)
 - [Modify Count Table Parameters](#)
- [Manipulation](#)
 - [Add Columns](#)
 - [Add Rows](#)
 - [Apply SQL Transformation](#)
 - [Clean Missing Data](#)
 - [Convert to Indicator Values](#)
 - [Edit Metadata](#)
 - [Group Categorical Values](#)
 - [Join Data](#)
 - [Remove Duplicate Rows](#)
 - [Select Columns in Dataset](#)
 - [Select Columns Transform](#)
 - [SMOTE](#)
- [Sample and Split](#)
 - [Partition and Sample](#)
 - [Split Data](#)
 - [Split Data using Split Rows](#)
 - [Split Data using Recommender Split](#)
 - [Split Data using Regular Expression](#)

[Split Data using Relative Expression](#)

[Scale and Reduce](#)

[Clip Values](#)

[Group Data into Bins](#)

[Normalize Data](#)

[Principal Component Analysis](#)

[Feature Selection](#)

[Filter Based Feature Selection](#)

[Fisher Linear Discriminant Analysis](#)

[Permutation Feature Importance](#)

[Machine Learning Modules](#)

[Evaluate](#)

[Cross-Validate Model](#)

[Evaluate Model](#)

[Evaluate Recommender](#)

[Initialize Model](#)

[Anomaly Detection](#)

[One-Class Support Vector Machine](#)

[PCA-Based Anomaly Detection](#)

[Classification](#)

[Multiclass Decision Forest](#)

[Multiclass Decision Jungle](#)

[Multiclass Logistic Regression](#)

[Multiclass Neural Network](#)

[One-vs-All Multiclass](#)

[Two-Class Averaged Perceptron](#)

[Two-Class Bayes Point Machine](#)

[Two-Class Boosted Decision Tree](#)

[Two-Class Decision Forest](#)

[Two-Class Decision Jungle](#)

[Two-Class Locally Deep Support Vector Machine](#)

[Two-Class Logistic Regression](#)

[Two-Class Neural Network](#)

[Two-Class Support Vector Machine](#)

[Clustering](#)

[K-Means Clustering](#)

[Regression](#)

[Bayesian Linear Regression](#)

[Boosted Decision Tree Regression](#)

[Decision Forest Regression](#)

[Fast Forest Quantile Regression](#)

[Linear Regression](#)

[Neural Network Regression](#)

[Ordinal Regression](#)

[Poisson Regression](#)

[Score](#)

[Apply Transformation](#)

[Assign Data to Clusters](#)

[Score Matchbox Recommender](#)

[Score Model](#)

[Train](#)

[Sweep Clustering](#)

[Train Anomaly Detection Model](#)

[Train Clustering Model](#)

[Train Matchbox Recommender](#)

[Train Model](#)

[Tune Model Hyperparameters](#)

[OpenCV Library Modules](#)

[Import Images](#)

[Pretrained Cascade Image Classification](#)

[Python Language Modules](#)

[Execute Python Script](#)

[R Language Modules](#)

[Execute R Script](#)

[Create R Model](#)

[R Packages Supported by Azure Machine Learning](#)

[Statistical Functions](#)

[Apply Math Operation](#)

[Compute Elementary Statistics](#)

[Compute Linear Correlation](#)

[Evaluate Probability Function](#)

[Replace Discrete Values](#)

[Summarize Data](#)

[Test Hypothesis Using t-Test](#)

[Text Analytics](#)

[Detect Languages](#)

[Extract Key Phrases from Text](#)

[Extract N-Gram Features from Text](#)

[Feature Hashing](#)

[Latent Dirichlet Allocation](#)

[Named Entity Recognition](#)

[Preprocess Text](#)

[Score Vowpal Wabbit 7-4 Model](#)

[Score Vowpal Wabbit 7-10 Model](#)

[Score Vowpal Wabbit 8 Model](#)

[Train Vowpal Wabbit 7-4 Model](#)

[Train Vowpal Wabbit 7-10 Model](#)

[Train Vowpal Wabbit 8 Model](#)

[Time Series](#)

[Time Series Anomaly Detection](#)

[Data Types](#)

[Data Table](#)

[ICluster interface](#)

[IFilter interface](#)

[ILearner interface](#)

[ITransform interface](#)

Module Error Codes

Azure Machine Learning Studio (classic): Algorithm and module help

3/10/2021 • 3 minutes to read • [Edit Online](#)

TIP

Customers currently using or evaluating Machine Learning Studio (classic) are encouraged to try [Azure Machine Learning designer](#), which provides drag-n-drop ML modules **plus** scalability, version control, and enterprise security.

Azure Machine Learning Studio (classic) is a cloud predictive analytics service that makes it possible to quickly create and deploy predictive models as analytics solutions. The machine learning tools are mostly cloud-based services, which eliminates setup and installation concerns because you can work through your web browser on any internet-connected PC. See the article, "[What is Studio \(classic\)?](#)" for more details.

This documentation contains detailed technical and how-to information for the modules that are available in Machine Learning Studio (classic).

- Sign in to your [Machine Learning Studio \(classic\) workspace](#) and get started.

What is a module?

Each *module* in Machine Learning Studio (classic) represents a set of code that can run independently and perform a machine learning task, given the required inputs. A module might contain a particular algorithm, or perform a task that is important in machine learning, such as missing value replacement, or statistical analysis.

In Studio (classic), modules are organized by functionality:

- **Data input and output modules** do the work of moving data from cloud sources into your experiment. You can write your results or intermediate data to Azure Storage, a SQL database, or Hive, while running an experiment, or use cloud storage to exchange data between experiments.
- **Data transformation modules** support operations on data that are unique to machine learning, such as normalizing or binning data, feature selection, and dimensionality reduction.
- **Machine learning algorithms**, such as clustering, support vector machine, or neural networks, are available within individual modules that let you customize the machine learning task with appropriate parameters. For classification tasks, you can choose from binary or multiclass algorithms.

After you've configured the model, use a [training module](#) to run data through the algorithm, and measure the accuracy of the trained model by using one of the [evaluation modules](#). To get predictions from the model you've just trained, use one of the [scoring modules](#).

- **Anomaly detection:** Machine Learning Studio (classic) includes multiple algorithms specialized for these tasks.
- **Text analytics modules** support various natural language processing tasks.
- **Vowpal Wabbit** support makes it easy to use this scalable platform.
- **Python** and **R language** modules make it easy to run a custom function. You write the code, and embed it in a module, to integrate Python and R with an experiment service.
- **OpenCV library** provides modules to use in specific image recognition tasks.

- [Time series analysis](#) supports anomaly detection in time series.
- [Statistical modules](#) provide a wide variety of numerical methods related to data science. Look in this group for correlation methods, data summaries, and statistical and math operations.

In this reference section, you'll find technical background on the machine learning algorithms, implementation details if available, and links to sample experiments that demonstrate how the module is used. You can download examples in the [Azure AI Gallery](#) to your workspace. These examples are for public use.

TIP

If you are signed in to Machine Learning Studio (classic) and have created an experiment, you can get information about a specific module. Select the module, then select the **more help** link in the **Quick Help** pane.

Other technical references

SECTION	DESCRIPTION
Data Types List	This section contains reference topics describing the learner interfaces, and the <code>DataTable</code> format used for datasets.
Exceptions List	This section lists the errors that modules can generate, with causes and possible workarounds. For the list of error codes related to the web service API, see Machine Learning REST API error codes .

See also

[Azure Machine Learning Studio \(classic\) documentation](#)

A-Z list of Machine Learning Studio (classic) modules

3/10/2021 • 8 minutes to read • [Edit Online](#)

This article provides an alphabetized list of the modules that are available in Azure Machine Learning Studio (classic).

TIP

Customers currently using or evaluating Machine Learning Studio (classic) are encouraged to try [Azure Machine Learning designer](#), which provides drag-n-drop ML modules **plus** scalability, version control, and enterprise security.

The modules cover a wide range of features and functions necessary for machine learning tasks:

- Data conversion functions
- Data transformation functions
- Modules for executing R or Python script
- Algorithms, including:
 - Decision trees
 - Decision forests
 - Clustering
 - Time series
 - Recommendation models
 - Anomaly detection

To find a module:

- If you know the name of the module, use the [alphabetical table](#) as an index to quickly find a specific module or algorithm.
- For a list of the modules by functional category, see [Module categories and descriptions](#).
- If you don't know which module you might need, see [Choose algorithms for Microsoft Azure Machine Learning](#).

Alphabetical table of modules

MODULE NAME	DESCRIPTION
Add Columns	Adds a set of columns from one dataset to another.
Add Rows	Appends a set of rows from an input dataset to the end of another dataset.
Apply Filter	Applies a filter to specified columns of a dataset.
Apply Math Operation	Applies a mathematical operation to column values.

MODULE NAME	DESCRIPTION
Apply SQL Transformation	Runs a SQLite query on input datasets to transform the data.
Apply Transformation	Applies a well-specified data transformation to a dataset.
Assign Data to Clusters	Assigns data to clusters by using an existing trained clustering model.
Bayesian Linear Regression	Creates a Bayesian linear regression model.
Boosted Decision Tree Regression	Creates a regression model by using the boosted decision tree algorithm.
Build Counting Transform	Creates counts to use to build features.
Clean Missing Data	Specifies how to handle values that are missing from a dataset.
Clip Values	Detects outliers, and then clips or replaces their values.
Compute Elementary Statistics	Calculates specified summary statistics for selected dataset columns.
Detect Languages	Detects the language of each line in the input file.
Compute Linear Correlation	Calculates the linear correlation between column values in a dataset.
Convert to ARFF	Converts data input to the attribute relation file format that's used by the Weka toolset.
Convert to CSV	Converts data input to a comma-separated values format.
Convert to Dataset	Converts data input to the internal dataset format that's used by Azure Machine Learning.
Convert to Indicator Values	Converts categorical values in columns to indicator values.
Convert to SVMLight	Converts data input to the format that's used by the SVMlight framework.
Convert to TSV	Converts data input to the tab-delimited format.
Create R Model	Creates an R model by using custom resources.
Cross-Validate Model	Cross-validates parameter estimates for classification or regression models by partitioning the data.
Decision Forest Regression	Creates a regression model by using the decision forest algorithm.
Detect Languages	Detects the language of each line in the input file.

MODULE NAME	DESCRIPTION
Edit Metadata	Edits metadata that's associated with columns in a dataset.
Enter Data Manually	Enables entering and editing small datasets by typing values.
Evaluate Model	Evaluates a scored classification or regression model by using standard metrics.
Evaluate Probability Function	Fits a specified probability distribution function to a dataset.
Evaluate Recommender	Evaluates the accuracy of recommender model predictions.
Execute Python Script	Executes a Python script from an Azure Machine Learning experiment.
Execute R Script	Executes an R script from an Azure Machine Learning experiment.
Export Count Table	Exports counts from a count transform.
Export Data	Writes a dataset to web URLs or to various forms of cloud-based storage in Azure, such as tables, blobs, and Azure SQL databases. This module was formerly named Writer .
Extract Key Phrases from Text	Extracts key words and phrases from a text column.
Extract N-Gram Features from Text	Creates N-Gram dictionary features, and then does feature selection on them.
Fast Forest Quantile Regression	Creates a quantile regression model.
Feature Hashing	Converts text data to integer-encoded features by using the Vowpal Wabbit library.
Filter Based Feature Selection	Identifies the features in a dataset that have the greatest predictive power.
FIR Filter	Creates a finite impulse response filter for signal processing.
Fisher Linear Discriminant Analysis	Identifies the linear combination of feature variables that can best group data into separate classes.
Group Categorical Values	Groups data from multiple categories into a new category.
Group Data into Bins	Puts numerical data into bins.
IIR Filter	Creates an infinite impulse response filter for signal processing.
Import Count Table	Imports counts from an existing count table.

MODULE NAME	DESCRIPTION
Import Data	Loads data from external sources on the web or from various forms of cloud-based storage in Azure, such as tables, blobs, SQL databases, and Azure Cosmos DB. Can load data from an on-premises SQL Server database if a gateway has been configured. This module was formerly named Reader .
Import Images	Loads images from Azure Blob storage into a dataset.
Join Data	Joins two datasets.
K-Means Clustering	Configures and initializes a K-means clustering model.
Latent Dirichlet Allocation	Performs topic modeling by using the Vowpal Wabbit library for Latent Dirichlet Allocation (LDA).
Linear Regression	Creates a linear regression model.
Load Trained Model	Gets a trained model that you can use for scoring in an experiment.
Median Filter	Creates a median filter that's used to smooth data for trend analysis.
Merge Count Transform	Merges two sets of count tables.
Modify Count Table Parameters	Builds a compact set of count-based features from count tables.
Moving Average Filter	Creates a moving average filter that smooths data for trend analysis.
Multiclass Decision Forest	Creates a multiclass classification model by using the decision forest algorithm.
Multiclass Decision Jungle	Creates a multiclass classification model by using the decision jungle algorithm.
Multiclass Logistic Regression	Creates a multiclass logistic regression classification model.
Multiclass Neural Network	Creates a multiclass classification model by using a neural network algorithm.
Named Entity Recognition	Recognizes named entities in a text column.
Neural Network Regression	Creates a regression model by using a neural network algorithm.
Normalize Data	Rescales numeric data to constrain dataset values to a standard range.

MODULE NAME	DESCRIPTION
One-Class Support Vector Machine	Creates a one-class support vector machine model for anomaly detection.
One-vs-All Multiclass	Creates a multiclass classification model from an ensemble of binary classification models.
Ordinal Regression	Creates an ordinal regression model.
Partition and Sample	Creates multiple partitions of a dataset based on sampling.
Permutation Feature Importance	Computes the permutation feature importance scores of feature variables in a trained model and a test dataset.
PCA-Based Anomaly Detection	Creates an anomaly detection model by using Principal Component Analysis (PCA).
Poisson Regression	Creates a regression model that assumes data has a Poisson distribution.
Preprocess Text	Performs cleaning operations on text.
Pretrained Cascade Image Classification	Creates a pretrained image classification model for frontal faces by using the OpenCV Library.
Principal Component Analysis	Computes a set of features that have reduced dimensionality for more efficient learning.
Remove Duplicate Rows	Removes duplicate rows from a dataset.
Replace Discrete Values	Replaces discrete values from one column with numeric values based on another column.
Score Matchbox Recommender	Scores predictions for a dataset by using the Matchbox recommender.
Score Model	Scores predictions for a trained classification or regression model.
Score Vowpal Wabbit 7-4 Model	<p>Scores data by using the Vowpal Wabbit machine learning system.</p> <p>Requires a trained model built by using Vowpal Wabbit versions 7-4 and 7-6.</p>
Score Vowpal Wabbit 7-10 Model	<p>Scores data by using the Vowpal Wabbit machine learning system.</p> <p>Requires a trained model built by using Vowpal Wabbit version 7-10.</p>

MODULE NAME	DESCRIPTION
Score Vowpal Wabbit 8 Model	Scores data by using the Vowpal Wabbit machine learning system from the command-line interface. Requires a trained model built by using Vowpal Wabbit version 8.
Select Columns in Dataset	Selects columns to include in or exclude from a dataset in an operation.
SMOTE	Increases the number of low-incidence examples in a dataset by using synthetic minority oversampling.
Split Data	Partitions the rows of a dataset into two distinct sets.
Summarize Data	Generates a basic descriptive statistics report for the columns in a dataset.
Sweep Clustering	Performs a parameter sweep on a clustering model to determine the optimum parameter settings.
Test Hypothesis Using T-Test	Compares means from two datasets by using a t-test.
Threshold Filter	Creates a threshold filter that constrains values.
Time Series Anomaly Detection	Learns a trend in time series data, and then uses the trend to detect anomalies.
Train Anomaly Detection Model	Trains an anomaly detector model, and then labels data from the training set.
Train Clustering Model	Trains a clustering model, and then assigns data from the training set to clusters.
Train Matchbox Recommender	Trains a Bayesian recommender by using the Matchbox algorithm.
Train Model	Trains a classification or regression model in a supervised manner.
Train Vowpal Wabbit 7-4 Model	Trains a model from the Vowpal Wabbit machine learning system. This module is for compatibility with Vowpal Wabbit versions 7-4 and 7-6.
Train Vowpal Wabbit 7-10 Model	Trains a model from the Vowpal Wabbit machine learning system. This module is for Vowpal Wabbit version 7-10.
Train Vowpal Wabbit 8 Model	Trains a model by using version 8 of the Vowpal Wabbit machine learning system. This module is for Vowpal Wabbit version 8.

MODULE NAME	DESCRIPTION
Tune Model Hyperparameters	Performs a parameter sweep on a regression or classification model to determine the optimum parameter settings.
Two-Class Averaged Perceptron	Creates an averaged perceptron binary classification model.
Two-Class Bayes Point Machine	Creates a Bayes point machine binary classification model.
Two-Class Boosted Decision Tree	Creates a binary classifier by using a boosted decision tree algorithm.
Two-Class Decision Forest	Creates a two-class classification model by using the decision forest algorithm.
Two-Class Decision Jungle	Creates a two-class classification model by using the decision jungle algorithm.
Two-Class Locally Deep Support Vector Machine	Creates a binary classification model by using the locally deep support vector machine algorithm.
Two-Class Logistic Regression	Creates a two-class logistic regression model.
Two-Class Neural Network	Creates a binary classifier by using a neural network algorithm.
Two-Class Support Vector Machine	Creates a binary classification model by using the support vector machine algorithm.
Unpack Zipped Datasets	Unpacks datasets from a .zip package in user storage.
User-Defined Filter	Creates a custom finite or infinite impulse response filter.

See also

- [Module categories and descriptions](#)
- [Module data types](#)

Machine Learning module descriptions

3/10/2021 • 4 minutes to read • [Edit Online](#)

This topic provides an overview of all the *modules* included in Azure Machine Learning Studio (classic), which is an interactive, visual workspace to easily build and test predictive models.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

What is a module?

In Machine Learning Studio (classic), a module is a building block for creating experiments. Each module encapsulates a specific machine learning algorithm, function, or code library that can act on data in your workspace. The modules are designed to accept connections from other modules, to share and modify data.

The code that runs in each module comes from many sources. These include open source libraries and languages, algorithms developed by Microsoft Research, and tools for working with Azure and other cloud services.

TIP

Looking for machine learning algorithms? See the [Machine Learning](#) category, which contains modules for decision trees, clustering, neural networks, among others. The [Train](#) and [Evaluate](#) categories include modules to help train and test your models.

By connecting and configuring modules, you can create a workflow that reads data from external sources, prepares it for analysis, applies machine learning algorithms, and generates results.

When an *experiment* is open in Machine Learning Studio (classic), you can see the complete list of current modules in the navigation pane at left. You drag these building blocks into your experiment, and then connect them to create a complete machine learning workflow, called an experiment.

Sometimes modules are updated to add new functionality, or to remove older code. When this happens, any experiments that you created that use the module continue to run. But the next time you open the experiment, you are prompted to upgrade the module, or to use a different module.

Examples

For an example of how to build a complete machine learning experiment, see these tutorials:

- [Develop a predictive solution by using Azure Machine Learning](#)
- [Create a simple experiment in Azure Machine Learning Studio \(classic\)](#)

Module categories

To make it easier to find related modules, the machine learning tools in Machine Learning Studio (classic) are grouped by these categories.

Data Format conversions

Use these modules to convert data to one of the formats used by other machine learning tools or formats.

- [Data Input and Output](#)

Use these modules to read data and models from cloud data sources, including Hadoop clusters, Azure Table storage, and web URLs. You can also use these modules to write results to storage or to a database.

- [Data Transformation](#)

Use these modules to prepare data for analysis. You can change data types, flag columns as features or labels, generate features, and scale or normalize data.

- [Filter](#)

Transform numeric data derived from digital signal processing.

- [Learning With Counts](#)

Use joint probability distributions to build features that compactly describe large datasets.

- [Manipulation](#)

This group provides a variety of tools for data science. For example, you can remove or replace missing values, choose a subset of columns, add a column, or concatenate two datasets.

- [Sample and Split](#)

Divide a dataset by criteria or by size, to create training and test sets, or to isolate certain rows.

- [Scale and Reduce](#)

Transform numerical data.

Feature Selection

Use these modules to identify the best features in your data, using widely researched statistical methods.

Machine Learning

This group contains most of the machine learning algorithms supported by Machine Learning.

It also contains modules intended to support the algorithms by training models, generating scores, and evaluating model performance.

- [Evaluate](#)

After you have trained a model, use these tools to measure the model's accuracy.

- [Initialize](#)

These modules provide the machine learning algorithms, which you can customize by setting parameters.

The algorithms in this section are grouped by type:

- [Anomaly detection algorithms](#)
 - [Classification algorithms](#)
 - [Clustering algorithms](#)
 - [Regression algorithms](#)
- [Score](#)

Use these modules to pass new data through the algorithm, and generate a set of results for evaluation.

You can also use the results of scoring as part of a predictive service.

- [Train](#)

These modules train an initialized machine learning model on data you provide.

[OpenCV Library Modules](#)

These modules give you easy access to a popular open source library for image processing and image classification.

[R Language Modules](#)

Use these modules to add custom R code to your experiment, or implement a machine learning model based on an R package.

[Python Language Modules](#)

Use these modules to add custom Python code to your experiment.

[Statistical Functions](#)

Use these modules to calculate probability distributions, create custom calculations, and perform a wide variety of other tasks related to numerical variables.

[Text Analytics](#)

Use these modules to perform feature hashing and named entity recognition, or to preprocess text using natural language processing tools.

[Time Series](#)

Use these modules to assess anomalies in trends, by using algorithms specifically designed for time series data.

Related tasks

Machine Learning Studio (classic) modules don't attempt to duplicate data integration tools supported in other tools, such as Azure Data Factory. Instead, the modules provide functionality that is specific to machine learning:

- Normalization, grouping, and scaling of data
- Computing statistical distribution of data
- Conversion to other machine learning formats
- Import of data used for machine learning experiments and export of results
- Text analytics, feature selection, and dimensionality reduction

If you need more sophisticated facilities for data manipulation and storage, see the following:

- [Azure Data Factory](#): Enterprise-ready, cloud data processing pipelines.
- [Azure SQL Database](#): Scalable storage, with integrated access to machine learning.
- [CosmosDB](#): NoSQL data store; import data to Machine Learning Studio (classic).
- [Azure Data Lake Analytics](#): Distributed analytics on big data.
- [Stream Analytics](#): Event processing for the Internet of Things.
- [Azure Text Analytics](#): Multiple options for text processing, and related cognitive services for speech, image, and facial recognition.
- [Azure Databricks](#): Spark-based analytics platform.

See also

- [A-Z module list](#)

Data Format Conversions

3/10/2021 • 3 minutes to read • [Edit Online](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

This article lists the modules provided in Azure Machine Learning Studio (classic) for converting data among various file formats used in machine learning.

The supported formats include:

- The **dataset** format that's used throughout Azure Machine Learning.
- The **ARFF** format that's used by [Weka](#). Weka is an open-source Java-based set of machine learning algorithms.
- The **SVMLight** format. The SVMLight format was developed for the [SVMLight](#) framework for machine learning. It can also be used by Vowpal Wabbit.
- The **tab-separated (TSV)** and **comma-separated (CSV)** flat file formats that are supported by most relational databases. These formats are also widely supported by R and Python.

When you convert data to these formats, you can more easily move results and data between different machine learning frameworks or storage mechanisms.

NOTE

These data conversion modules only convert the complete dataset to a specified format. If you need to do any casting, truncation, conversion of date-time formats, or other manipulation of the values, use the modules in [Data Transformation](#), or see the list of [related tasks](#).

Common data conversion scenarios

You typically use the data conversion modules if you need to move data from an Azure Machine Learning experiment to another machine learning tool or platform. You also can use the modules to export data from Machine Learning in a format that can be used by a database or other tools. For example:

TASK	USE THIS
You need to save an intermediate dataset to use in Excel, or to import to a database.	Use the CSV module or the TSV module to prepare the data in the correct format. Then, either download the data or save it to Azure Storage.
You want to reuse data from your experiment in R or Python code.	Use the CSV module or the TSV module to prepare the data. Then, right-click the converted dataset to get the Python code that you need to access the dataset.
You are porting your experiment and data between Weka and Azure Machine Learning.	Use the ARFF module to prepare the data. Then, download the results.

TASK	USE THIS
You need to prepare data in the SVMlight framework.	Use the Convert to SVMLight module to prepare the data. Then, download the resulting data.
Create data to use with Vowpal Wabbit.	Use the SVMLight format. Then, modify the files as described in the article. Save the file in Azure Blob storage to use with a Vowpal Wabbit module in Azure Machine Learning.
Data is not in a tabular format.	Coerce it to a dataset format by using the Convert to Dataset module.

Related tasks

If you need to import data into Azure Machine Learning or transform data in individual columns, use these modules before you perform data conversion:

TASK	USE THIS
Import data from my computer into Azure Machine Learning.	Upload datasets in CSV format as described in Import your training data into Azure Machine Learning Studio (classic) .
Import data from a cloud data source, including Hadoop or Azure.	Use the Import Data module.
Save machine learning datasets to Azure Blob storage, a Hadoop cluster, or other cloud-based storage.	Use the Export Data module.
Change the data type of columns or cast columns to a different format or type.	In Azure Machine Learning, use the Edit Metadata or Apply SQL Transformation modules. If you are proficient with R or Python, try the Execute Python Script or Execute R Script modules.
Round, group, or normalize numerical data.	Use the Apply Math Operation , Group Data into Bins , or Normalize Data modules.

List of modules

The **Data Format Conversions** category includes these modules:

- [Convert to ARFF](#): Converts data input to the attribute relation file format that's used by the Weka toolset.
- [Convert to CSV](#): Converts a dataset to a comma-separated values format.
- [Convert to Dataset](#): Converts data input to the internal dataset format that's used by Azure Machine Learning.
- [Convert to SVMLight](#): Converts data input to the format that's used by the SVMlight framework.
- [Convert to TSV](#): Converts data input to the tab-delimited format.

See also

- [Data Transformation](#)
- [Module categories and descriptions](#)

Convert to ARFF

3/10/2021 • 3 minutes to read • [Edit Online](#)

Converts data input to the attribute relation file format used by the Weka toolset

Category: [Data Format Conversions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Convert to ARFF](#) module in Azure Machine Learning Studio (classic), to convert datasets and results in Azure Machine Learning to the attribute-relation file format used by the Weka toolset. This format is known as ARFF.

The ARFF data specification for Weka supports multiple machine learning tasks, including data preprocessing, classification, and feature selection. In this format, data is organized by entities and their attributes, and is contained in a single text file. You can find details of the Weka file format in the [Technical Notes](#) section.

In general, conversion to the Weka file format is required only if you want to use both Azure Machine Learning and Weka, and intend to move your training data back and forth between them.

For more information about the Weka toolset, see this Wikipedia article: [Weka \(machine learning\)](#)

WARNING

You cannot overwrite an existing ARFF file in Azure Storage.

How to use Convert to ARFF

1. Add the [Convert to ARFF](#) module to your experiment. You can find this module in the [Data Format Conversions](#) category in Azure Machine Learning Studio (classic).
2. Connect it to any module that outputs a dataset.
3. Run the experiment, or click the [Convert to ARFF](#) module, and click **Run selected**.

Results

- To create a copy of the data in a local folder, double-click the output of [Convert to ARFF](#), and select the **Download** option.

If you do not specify a folder, a default file name is applied and the file is saved in the local **Downloads** library.

NOTE

This module does not support export to Python or R code.

Examples

There are no examples specific to this format in the [Azure AI Gallery](#). However, these experiments demonstrate other types of format conversion:

- [Color-Based Image Compression](#): Exports the datasets used for each portion of the analysis to files for reproducibility and use on other analytics platforms.
- [Cross Validation for Binary Classification sample](#): Exports the results of cross validation to files so that the results for multiple models can be compared by using a tool such as Excel.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Example of ARFF format

This section provides an example of how a typical dataset would look when converted to ARFF.

Typically an ARFF data file is comprised of two sections: a **header** that defines the data source and schema, and the **data** section, which contains the actual entities and their attributes.

ARFF header

The header for an ARFF file defines the list of the attributes (in columns) and their data types. The header can also contain multiple comment lines that describe the data source or any other notes.

```
% Source: Iris dataset, UCI % 0 = Iris-setosa, 1= Iris-virginica @RELATION iris @ATTRIBUTE sepal_length  
NUMERIC @ATTRIBUTE sepal_width NUMERIC @ATTRIBUTE petal_length NUMERIC @ATTRIBUTE petal_width NUMERIC  
@ATTRIBUTE class {0, 1}
```

TIP

If the dataset you are converting does not have column names, use the [Edit Metadata](#) module to add column names before using converting to ARFF.

ARFF data

The data section consists of comma-separated values, and looks very much like a CSV file without column headings.

```
@DATA 5.1,3.5,1.4,0.2,0
```

For additional information about this file format, see the Weka Wiki page: [ARFF \(developer version\)](#).

Current ARFF version

Azure Machine Learning Studio (classic) saves ARFF files by using the ARFF 3.0 format.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Arff	Output dataset

See also

[Data Format Conversions](#)

[A-Z Module List](#)

Convert to CSV

3/10/2021 • 6 minutes to read • [Edit Online](#)

Converts data input to a comma-separated values format

Category: [Data Format Conversions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Convert to CSV](#) module in Azure Machine Learning Studio (classic), to convert a dataset from Azure ML into a CSV format that can be downloaded, exported, or shared with R or Python script modules.

More about the CSV format

The CSV format, which stands for "comma-separated values", is a file format used by many external machine learning tools. Although the native dataset format used by Azure Machine Learning is based on the .NET datatable and thus can be read by .NET libraries, CSV is a common interchange format when working with open-source languages such as R or Python.

Even if you do most of your work in Azure Machine Learning Studio (classic), there are times when you might find it handy to convert your dataset to CSV to use in external tools. For example:

- Download the CSV file to open it with Excel, or import it into a relational database.
- Save the CSV file to cloud storage and connect to it from Power BI to create visualizations.
- Use the CSV format to prepare data for use in R and Python. Just right-click the output of the module to generate the code needed to access the data directly from Python or a Jupyter notebook.

When you convert a dataset to CSV, the file is saved in your Azure ML workspace. You can use an Azure storage utility to open and use the file directly, or you can right-click the module output and download the CSV file to your computer, or use it in R or Python code.

How to configure Convert to CSV

1. Add the [Convert to CSV](#) module to your experiment. You can find this module in the [Data Format Conversions](#) group in Studio (classic).
2. Connect it to any module that outputs a dataset.
3. Run the experiment, or click the [Convert to CSV](#) module, and click **Run selected**.

Results

Double-click the output of [Convert to CSV](#), and select one of these options.

- **Download:** Immediately opens a copy of the data in CSV format that you can save to a local folder. If you do not specify a folder, a default file name is applied and the CSV file is saved in the local **Downloads**

library.

If you select **Download dataset**, you must indicate whether you want to open the dataset, or save it to a local file.

If you select **Open**, the dataset is loaded using the application that is associated by default with .CSV files: for example, Microsoft Excel.

If you select **Download dataset**, by default, the file is saved with the name of the module plus a GUID representing the workspace ID. However, you can select the **Save As** option during download and change the file name or location.

- **Save as Dataset:** Saves the CSV file back to the Azure ML workspace as a separate dataset.
- **Generate Data Access Code:** Azure ML generates two sets of code for you to access the data, either by using Python or by using R. To access the data, copy the code snippet into your application.
- **Open in a new Notebook:** A new Jupyter notebook is created for you and code inserted for reading the data from your workspace, using the language of your choice: Python 2, Python 3, or R with Microsoft R Open.

For example, if you choose the R option, sample R code is provided that loads the CSV file into a data frame and displays the first few rows using the `head` function.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Requirements of the CSV format

The CSV file format is a popular format supported by many machine learning frameworks. The format is variously referred to "comma-separated values" or "character-separated values."

A CSV file stores tabular data (numbers and text) in plain text form. A CSV file consists of any number of records, separated by line breaks of some kind. Each record consists of fields, separated by a literal comma. In some regions, the separator might be a semi-colon.

Typically, all records have an identical number of fields, and missing values are represented as nulls or empty strings.

TIP

You can easily export data from Excel, Access, or a relational database into CSV files, to use in Azure Machine Learning. Although file names typically have the .CSV extension, Azure Machine Learning does not require that this filename extension be present if you want to import the data as CSV. You can import XLSX, TXT, and other files as CSV. However, the fields in the file must be formatted as described in the preceding section, and the file must use the UTF-8 encoding.

Common questions and issues

This section describes some known issues, common questions, and workarounds specific to the [Convert to CSV](#) module.

Headers must be single rows

The CSV file format used in Azure Machine Learning supports a single header row. You cannot insert multi-line headers.

Custom separators supported on import but not export

The [Convert to CSV](#) module does not support generating alternative column separators, such as the semicolon (;), which is often used in Europe.

However, when you import data from CSV files in external storage, you can specify alternative separators. In the [Import Data](#) module, select the **CSV with encodings** option, and pick a supported encoding.

Inaccurate column separation on string data containing commas

It is a common problem in text processing that just about every character that can be specified as a column separator (tabs, spaces, commas, etc.) can also be found randomly in text fields. Importing text from CSV always requires caution to avoid separating text across unnecessary new columns.

When you try to export a column of string data that contains commas, you might run into problems as well. Azure Machine Learning does not support any special handling or special translation of such data, such as enclosing strings in quotation marks. Also, you cannot use escape characters before a comma to ensure that commas are handled as a literal character.

Therefore, new fields are created in the output file for each comma that is encountered in the string field. To avoid this problem, there are several workarounds:

- Use the [Preprocess Text](#) module to remove punctuation characters from string fields.
- Use custom [R script](#) or [Python script](#) to process text and ensure that data can be exported correctly.

UTF-8 encoding required

The [Convert to CSV](#) module supports only UTF-8 character encoding. If you need to export data using a different encoding, you can try using the [Execute R Script](#) or [Execute Python Script](#) modules to generate custom output.

- [Encodings and R](#)
- [Python standard encodings](#)

Dataset does not have column names

If the dataset you are exporting to a CSV file does not have column names, we recommend that you use [Edit Metadata](#) to add column names before converting it. You cannot add column names as part of the conversion or export process.

SYLK: File format is not valid

If the first column of the dataset that you convert to CSV has the name ID, you might get the following error when you try to open the file in Excel:

"SYLK: File format is not valid."

To avoid this error, you must rename the column. For more information, see

<https://support.microsoft.com/kb/215591>

I need help with importing from CSV

For importing, don't use the [Export to CSV](#) module. Instead, use the [Import Data](#) module.

For general information about importing from CSV, see these resources:

- [Import your training data into Azure Machine Learning Studio \(classic\) from various data sources](#)
- [AzureML Experiments and Data Interaction](#): Demonstrates various data sources and how to work with them in Studio (classic).

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Output

NAME	TYPE	DESCRIPTION
Results dataset	GenericCsv	Output dataset

See also

[Data Format Conversions](#)

[A-Z Module List](#)

Convert to Dataset

3/10/2021 • 4 minutes to read • [Edit Online](#)

Converts data input to the internal Dataset format used by Microsoft Azure Machine Learning

Category: [Data Format Conversions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Convert to Dataset](#) module in Azure Machine Learning Studio (classic), to convert any data that you might need for an experiment to the internal format used by Studio (classic).

Conversion is not required in most cases, because Azure Machine Learning implicitly converts data to its native dataset format when any operation is performed on the data.

However, saving data to the dataset format is recommended if you have performed some kind of normalization or cleaning on a set of data, and you want to ensure that the changes are used in further experiments.

NOTE

[Convert to Dataset](#) changes only the format of the data, and it does not save a new copy of the data in the workspace. To save the dataset, double-click the output port, select **Save as dataset**, and type a new name.

How to use Convert to Dataset

We recommend that you use the [Edit Metadata](#) module to prepare the dataset before using [Convert to Dataset](#). You can add or change column names, adjust data types, and so forth.

1. Add the [Convert to Dataset](#) module to your experiment. You can find this module in the [Data Format Conversions](#) category in Azure Machine Learning Studio (classic).
2. Connect it to any module that outputs a dataset.

As long as the data is tabular, you can convert it to a dataset. This includes data loaded using [Import Data](#), data created by using [Enter Data Manually](#), data generated by code in custom modules, datasets transformed by using [Apply Transformation](#), or datasets that were generated or modified by using [Apply SQL Transformation](#).

3. In the **Action** dropdown list, indicate if you want to do any cleanup on the data before saving the dataset:
 - **None:** Use the data as is.
 - **SetMissingValue:** Specify a placeholder that is inserted in the dataset wherever there is a missing value. The default placeholder is the question mark character (?), but you can use the **Custom missing value** option to type a different value.

- **ReplaceValues**: Use this option to specify a single exact value to be replaced with any other exact value. For example, assuming your data contains the string `obs` used as a placeholder for missing values, you could specify a custom replacement operation using these options:
 - Set **Replace** to **Custom**
 - For **Custom value**, type the value you want to find. In this case, you would type `obs`.
 - For **New value**, type the new value to replace the original string with. In this case, you might type `?`

Note that the **ReplaceValues** operation applies only to exact matches. For example, these strings would not be affected: `obs.`, `obsolete`.

- **SparseOutput**: Indicates that the dataset is sparse. By creating a sparse data vector, you can ensure that missing values do not affect a sparse data distribution. After choosing this option, you must indicate how missing values and zero values should be handled.

To remove any value other than zero, click the **Remove** option and type a single value to remove. You can remove missing values, or set a custom value to delete from the vector. Only exact matches will be removed. For example, if you type `x` in the **Remove value** text box, the row `xx` would not be affected.

By default, the option **Remove zeroes** is set to `True`, meaning that all zero values are removed when the sparse column is created.

- Run the experiment, or right-click the [Convert to Dataset](#) module and select **Run selected**.

Results

- To save the resulting dataset with a new name, right-click the output of [Convert to Dataset](#) and select **Save as Dataset**.

Examples

You can see examples of how the [Convert to Dataset](#) module is used in the Azure AI Gallery:

- [CRM sample](#): Reads from a shared dataset and saves a copy of the dataset in the local workspace.
- [Flight Delay example](#): Saves a dataset that has been cleaned by replacing missing values so that you can use it for future experiments.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

- Any module that takes a dataset as input can also take data in the CSV, TSV, or ARFF formats. Before any module code is executed, preprocessing of the inputs is performed, which is equivalent to running the [Convert to Dataset](#) module on the input.
- You cannot convert from the SVMLight format to dataset.
- When specifying a custom replace operation, the search and replace operation applies to complete values; partial matches are not allowed. For example, you can replace a 3 with a -1 or with 33, but you cannot replace a 3 in a two-digit number such as 35.
- For custom replace operations, the replacement will silently fail if you use as a replacement any character that does not conform to the current data type of the column.
- If you need to save data that uses numerical data that is sparse and has missing values, internally, Studio

(classic) supports sparse arrays by using a `SparseVector`, which is a class in the Math.NET numeric library. Prepare your data that uses zeros and has missing values, and then use [Convert to Dataset](#) with the arguments `SparseOutput` and `Remove Zeros = TRUE`.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Action	List	Action Method	None	Action to apply to input dataset

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Output dataset

See also

[Data Format Conversions](#)

[A-Z Module List](#)

Convert to SVMLight

3/10/2021 • 5 minutes to read • [Edit Online](#)

Converts data input to the format used by the SVM-Light framework

Category: [Data Format Conversions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Convert to SVMLight](#) module in Azure Machine Learning Studio (classic), to convert your datasets to the format that is used by SVMLight.

The SVM-Light framework was developed by researchers at Cornell University. The SVM-Light library implements Vapnik's Support Vector Machine, but the format has been adopted elsewhere and can be used for many machine learning tasks, including classification and regression.

For more information, see [SVMLight Support Vector Machine](#).

How to configure Convert to SVMLight

The conversion to SVMLight format entails converting each case into a row of data that begins with the label, followed by feature-value pairs expressed as colon-separated numbers. The conversion process **does not** automatically identify the correct columns, so it is important that you prepare the columns in your dataset before attempting conversion. For more information, see [Preparing Data for Conversion](#).

1. Add the [Convert to SVMLight](#) module to your experiment. You can find this module in the [Data Format Conversions](#) category in Azure Machine Learning Studio (classic).
2. Connect the dataset or output that you want to convert to SVMLight format.
3. Run the experiment.
4. Right-click the output of the module, select **Download**, and save the data to a local file for modification or for reuse with a program that supports SVMLight.

Preparing data for conversion

To illustrate the conversion process, this example uses the **Blood Donor** dataset in Studio (classic).

This sample dataset has the following format, in tabular form.

RECENCY	FREQUENCY	MONETARY	TIME	CLASS
2	50	12500	98	1
0	13	3250	28	1

RECENCY	FREQUENCY	MONETARY	TIME	CLASS
1	1	4000	35	1
2	20	5000	45	1
1	24	6000	77	0

Note that the label column, named [Class] in this dataset, is the last column in the table. However, if you convert the dataset to SVMLight without first indicating that which column contains the label, the first column, [Recency], is used as the label, and the [Class] column is treated as a feature:

```
2 1:50 2:12500 3:98 4:1
0 1:13 2:3250 3:28 4:1
1 1:16 2:4000 3:35 4:1
```

To make sure the labels are generated correctly at the beginning of the row for each case, you must add two instances of the [Edit Metadata](#) module.

1. In the first instance of **Edit Metadata**, select the label column ([Class]) and for **Fields**, select **Label**.
2. In the second instance of **Edit Metadata**, select all feature columns that you need in the converted file ([Recency], [Frequency], [Monetary], [Time]) and for **Fields**, select **Features**.

After the columns have been identified correctly, you can run the **Convert to SVMLight** module. After conversion, the first few rows of the Blood Donor dataset now have this format:

- The label value precedes each entry, followed by the values for [Recency], [Frequency], [Monetary], and [Time], identified as features 1, 2, 3 and 4 respectively.
- The label value of 0 in the fifth row has been converted to -1. This is because SVMLight supports only binary classification labels.

```
1 1:2 2:50 3:12500 4:98
1 1:0 2:13 3:3250 4:28
1 1:1 2:16 3:4000 4:35
1 1:2 2:20 3:5000 4:45
-1 1:1 2:24 3:6000 4:77
```

You can't directly use this text data for models in Azure ML, or visualize it. However, you can download it to a local share.

While you have the file open, we recommend that you add a comment line, prefaced by `#`, so that you can add notes about the source or the original feature column names.

To use an SVMLight file in Vowpal Wabbit, and make additional modifications as described here: [Conversion to Vowpal Wabbit Format](#). When the file is ready, upload it to Azure blob storage, and call it directly from one of the Vowpal Wabbit modules.

Examples

There are no examples in the [Azure AI Gallery](#): that are specific to this format.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Usage tips

The executables provided in the SVM-Light framework require both an *example file* and a *model file*. However, this module creates only the example file. You must create the model file separately by using the SVMLight libraries.

The example file is the file that contains the training examples.

- **Optional header**

The first lines can contain comments. Comments must be prefixed with the number sign (#).

The file format output by **Convert to SVMLight** does not create headers. You can edit the file to add comments, a list of column names, and so forth.

- **Training data**

Each case is on its own row. A case consists of a target value followed by a series of indices and the associated feature values.

The response value must be 1 or -1 for classification, or a number for regression.

The target value and each of the index-value pairs are separated by a space.

Training data example

The following table shows how the values in the columns of the Two-Class Iris dataset are converted to a representation in which each column is represented by an index, followed by a colon, and then the value in that column:

IRIS DATASET	IRIS DATASET CONVERTED TO SVMLIGHT
1 6.3 2.9 5.6 1.8	1 1:6.3 2:2.9 3:5.6 4:1.8
0 4.8 3.4 1.6 0.2	-1 1:4.8 2:3.4 3:1.6 4:0.2
1 7.2 3.2 6 1.8	1 1:7.2 2:3.2 3:6 4:1.8

Note that the names of the feature columns are lost in the conversion.

Using SVMLight to prepare a Vowpal Wabbit file

The SVMLight format is similar to the format used by Vowpal Wabbit. To change the SVMLight output file to a format usable for training a Vowpal Wabbit model, just add a pipe symbol between the label and the list of features.

For example, compare these lines of input:

Vowpal Wabbit format, including optional comment

```
# features are [Recency], [Frequency], [Monetary], [Time]
1 | 1:2 2:50 3:12500 4:98
1 | 1:0 2:13 3:3250 4:28
```

SVMLight format, including optional comment

```
# features are [Recency], [Frequency], [Monetary], [Time]
1 1:2 2:50 3:12500 4:98
1 1:0 2:13 3:3250 4:28
```

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Output

NAME	TYPE	DESCRIPTION
Results dataset	SvmLight	Output dataset

See also

[Data Format Conversions](#)

[A-Z Module List](#)

Convert to TSV

3/10/2021 • 3 minutes to read • [Edit Online](#)

Converts data input to a tab-delimited format

Category: [Data Format Conversions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Convert to TSV](#) module in Azure Machine Learning Studio (classic), to convert any dataset from the internal format that is used by all Azure Machine Learning Studio (classic) modules, to a flat file in tab-separated format.

Tab-separated value (TSV) files are compatible with many external tools, including:

- R and Python
- Excel and PowerPivot
- All relational databases

For example, if your experiment has an intermediate dataset that you would like to save for re-use in another tool or would like to call from code, you convert it to the TSV format, and then right-click the converted dataset to get the Python code needed to access the dataset.

How to use Convert to TSV

Use the [Convert to TSV](#) module whenever you need to download a dataset in tab-delimited format.

1. Add the [Convert to TSV](#) to your experiment. You can find this module in the [Data Format Conversions](#) category in Azure Machine Learning Studio (classic).
2. Connect the module to another dataset, or to a module that outputs a tabular dataset.
3. Run the experiment, or right-click just the [Convert to TSV](#) module, and select **Run selected**.

Results

When conversion is complete, you can open the dataset, call it from R or Python code, use it in a Jupyter notebook, or save it to a local file.

If you want to download the dataset, double-click the module output, and indicate whether you want to open or save the dataset.

- If you select **Open**, the dataset is loaded using whatever tool your computer uses by default to open .TSV files. Typically this is Microsoft Excel.
- If you select **Download dataset**, by default, the file is saved with the name of the module plus a GUID

representing the workspace ID. However, you can select the **Save As** option during download and change the file name or location.

Examples

Although there are no examples that are specific to this format, you can see examples of how format conversion is used by exploring these sample experiments in the [Azure AI Gallery](#):

- [Cross Validation for Binary Classification sample](#): Exports the results of cross validation to the comma-separated value (CSV) format so that results for multiple models can be compared by using a tool such as Excel.
- [Color-Based Image Compression Quantization](#): Exports the datasets that are used for each portion of the analysis to CSV files, so that you can easily run a similar model in any tool that supports the CSV format.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

TSV format requirements

Tab-separated values (TSV) is a text format that is used to store data in a tabular structure. It is very similar to the CSV format, but the delimiter is a tab rather than a comma.

The TSV format is a useful alternative to the CSV format if your data contains commas. Commas are very common in text data and they are used in European number formats.

One problem with the tab-delimited format is that tab stops are frequently considered as white space in unstructured text. However, the IANA standard for TSV fosters clean and accurate parsing of TSV files by disallowing tabs within fields.

Note the following requirements for TSV files in Azure Machine Learning Studio (classic):

- The [Convert to TSV](#) module supports the output of a single heading row, if the dataset contains column names.
- The TSV provider supports UTF-8 character encoding only.
- When reading from or writing to TSV files, performance can be slower than with other formats (such as CSV).

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Output

NAME	TYPE	DESCRIPTION
Results dataset	GenericTsv	Output dataset

See also

[Data Format Conversions](#)

Data Input and Output

3/10/2021 • 2 minutes to read • [Edit Online](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

This article lists the modules that you can use for importing and exporting data and models in Azure Machine Learning Studio (classic).

In addition to using modules, you can directly upload and download datasets from local files on your computer or network. For more information, see [Upload existing data into an Azure Machine Learning experiment](#).

Here are some of the sources you can use to import and export data and models in Machine Learning Studio (classic):

- Get data from sources in the cloud, such as Azure SQL Database, Azure SQL Data Warehouse, Azure Storage, and Azure Cosmos DB. You can also import data that's provided as a public web URL, get data from Hadoop by using a Hive query, or query an on-premises SQL server.
- Load a collection of images from Azure Blob storage to use in image classification tasks.
- Extract the data from zipped files that you uploaded to Machine Learning. You can use the datasets in experiments.
- Create small datasets by typing in the Machine Learning Studio (classic) UI. This can be handy for creating small test datasets.
- Save your results or intermediate data to Azure Table storage, Blob storage, a SQL database, or a Hive query.
- Get a trained model from a URL or Blob storage, and then use it in an experiment.

NOTE

The modules in this group only move data to or from Machine Learning Studio (classic). You can't use the modules to filter, cast, or transform the data during the import or export process.

For more information about how to transform and filter your data in Machine Learning Studio (classic), see [Data Transformation](#).

Resources

The following articles introduce common data scenarios in machine learning:

Get started

- [Microsoft Machine Learning blog: The Cloud Data Science Process](#)

Learn how to manage data for machine learning in the cloud. The information in this article is based on CRISP-DM, an industry standard. The article provides end-to-end walkthroughs that demonstrate the integration of machine learning with cloud data solutions such as Azure HDInsight and SQL Database.

- [Import your training data into Machine Learning Studio \(classic\)](#)

This article describes how to get your data into Azure, and then create an experiment.

Advanced data science

- [Access datasets with Python by using the Azure Machine Learning Python client library](#)

Learn how to install the Machine Learning Python client library, and then use it to access metadata and work with datasets.

Sample experiments

- [Data processing](#)
- [Download data](#)

List of modules

The **Data Input and Output** category includes the following modules:

- [Enter Data Manually](#): Lets you create small datasets by typing values.
- [Export Data](#): Writes a dataset to web URLs or to various forms of cloud-based storage in Azure, such as tables, blobs, or a SQL database.
- [Import Data](#): Loads data from external sources on the web and from various forms of cloud-based storage in Azure, such as Table storage, Blob storage, SQL Database, SQL Data Warehouse, Azure Cosmos DB, or a Hive query. You can also import data from an on-premises SQL Server database.
- [Load Trained Model](#): Gets a trained model from a URL or Blob storage to use in a scoring experiment.
- [Unpack Zipped Datasets](#): Decompresses a dataset that's been stored in zipped format, and then adds the dataset to your workspace.

See also

- [Data Format Conversions](#)
- [Data Transformation](#)
- [Module categories and descriptions](#)

Enter Data Manually

3/10/2021 • 4 minutes to read • [Edit Online](#)

Enables entering and editing small datasets by typing values

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Enter Data Manually](#) module in Azure Machine Learning Studio (classic), to create a small dataset by typing values. The dataset can have multiple columns.

This module can be helpful in scenarios such as these:

- Generating a small set of values for testing
- Creating a short list of labels
- Entering values for use in [Apply Math Operation](#)
- Specifying replacement values for use in [Replace Discrete Values](#)
- Typing a list of column names to insert in a dataset

How to use Enter Data Manually

1. Add the [Enter Data Manually](#) module to your experiment. You can find this module in the [Data Input and Output](#) category in Azure Machine Learning Studio (classic).
2. For **DataFormat**, select one of the following options. These options determine how the data that you provide should be parsed. The requirements for each format differ greatly, so be sure to read the related topics.
 - **ARFF**. The attribute-relation file format, used by Weka. For more information, see [Convert to ARFF](#).
 - **CSV**. Comma-separated values format. For more information, see [Convert to CSV](#).
 - **SVMLight**. A format used by Vowpal Wabbit and other machine learning frameworks. For more information, see [Convert to SVMLight](#).
 - **TSV**. Tab-separated values format. For more information, see [Convert to TSV](#).If you choose a format and do not provide data that meets the format specifications, a run-time error occurs.
3. Click inside the **Data** text box to start entering data. The following formats require special attention:
 - **CSV**: To create multiple columns, paste in comma-separated text, or type multiple columns using

commas between fields.

If you select the **HasHeader** option, you can use the first row of values as the column heading.

If you deselect this option, the columns names, Col1, Col2 and so forth are used. You can add or change columns names later using [Edit Metadata](#).

- **TSV:** To create multiple columns, paste in tab-separated text, or type multiple columns using tabs between fields.

If you select the **HasHeader** option, you can use the first row of values as the column heading.

If you deselect this option, the columns names, Col1, Col2 and so forth are used. You can add or change columns names later using [Edit Metadata](#).

- **ARFF:** Paste in an existing ARFF format file. If you are typing values directly, be sure to add the optional header and required attribute fields at the beginning of the data.

For example, the following header and attribute rows could be added to a simple list. The column heading would be `SampleText`.

```
% Title: SampleText.ARFF
% Source: Enter Data module
@ATTRIBUTE SampleText STRING
@DATA
\<type first data row here>
```

- **SVMLight:** Type or paste in values using the SVMLight format.

For example, the following sample represents the first couple lines of the Blood Donation dataset, in SVMlight format:

```
# features are [Recency], [Frequency], [Monetary], [Time]
1 1:2 2:50 3:12500 4:98
1 1:0 2:13 3:3250 4:28
```

When you run the [Enter Data Manually](#) module, these lines are converted to a dataset of columns and index values as follows:

COL1	COL2	COL3	COL4	LABELS
0.00016	0.004	0.999961	0.00784	1
0	0.004	0.999955	0.008615	1

4. Press ENTER after each row, to start a new line.

Be sure to press ENTER after the final row.

If you press ENTER multiple times to add multiple empty trailing rows, the final empty row is removed trimmed, but other empty rows are treated as missing values.

If you create rows with missing values, you can always filter them out later.

5. Right-click the module and select **Run selected** to parse the data and load it into your workspace as a dataset.

To view the dataset, click the output port and select **Visualize**.

Examples

For examples of how this module is used in machine learning, see the [Azure AI Gallery](#):

- [Download Data sample](#): Gets data from the UCI Machine Learning repository and then uses [Enter Data Manually](#) to create column names. Sample R code is also provided, which you can use to merge the entered rows with the dataset.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

- Regardless of the saved format, data that you enter is implicitly converted to the dataset ([Data Table](#)) format for use in experiments. However, data is not persisted as a saved dataset unless you explicitly choose the **Save as Dataset** option.

If you do not save the data in [Enter Data Manually](#) as a dataset, it is removed from the workspace cache when you end the session. However, you can run the experiment again to make the data available.

- If you combine the data from [Enter Data Manually](#) with another dataset, the combined dataset cannot have two columns with the same name. If there are duplicate column names, a numeric suffix is appended to the column from the right dataset to make the column names unique.

For example, assume that you have two instances of [Enter Data Manually](#) that contain the column **TestData**, and use the [Add Columns](#) module to merge them. The column from the left instance of [Enter Data Manually](#) would remain as **TestData**, and the column from the right instance of [Enter Data Manually](#) would be renamed **TestData (2)**.

See also

[Data Input and Output](#)

[A-Z Module List](#)

Export Data

3/10/2021 • 7 minutes to read • [Edit Online](#)

Writes a dataset to various forms of cloud-based storage in Azure, such as tables, blobs, and Azure SQL databases

Category: [Data Input and Output](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Export Data](#) module in Azure Machine Learning Studio (classic), to save results, intermediate data, and working data from your experiments into cloud storage destinations outside Azure Machine Learning Studio (classic).

This module supports exporting or saving your data to the following cloud data services:

- [Export to Hive Query](#): Write data to a Hive table in an HDInsight Hadoop cluster.
- [Export to Azure SQL Database](#): Save data to Azure SQL Database or to Azure SQL Data Warehouse.
- [Export to Azure Table](#): Save data to the table storage service in Azure. Table storage is good for storing large amounts of data. It provides a tabular format that is scalable, inexpensive, and highly available.
- [Export to Azure Blob Storage](#): Saves data to the Blob service in Azure. This option is useful for images, unstructured text, or binary data. Data in the Blob service can be shared publicly or saved in secured application data stores.

NOTE

Export data module does not support connecting to Azure Blob storage account if "Secure Transfer Required" option is enabled.

Related tasks

- **Download data:** To download your data so that you can open it in Excel or another application, use a module such as [Convert to CSV](#) or [Convert to TSV](#) to prepare the data in a particular format, and then download the data.
- You can download the results of any module that outputs a dataset by right-clicking the output and selecting **Download dataset**. By default, the data is exported in CSV format.
- **Download a module definition or experiment graph:** A new PowerShell library lets you download the complete metadata for your experiment, or the details for a particular module. The PowerShell for Azure Machine Learning library is an experimental release, but has many useful cmdlets:
 - `Get-AmlExperiment` lists all the experiments in a workspace.

- `Export-AmlExperimentGraph` exports a definition of the complete experiment to a JSON file.
- `Download-AmlExperimentNodeOutput` lets you extract the information provided on the output ports of any module.

For more information, see [PowerShell Module for Azure Machine Learning Studio \(classic\)](#).

How to configure Export Data

1. Add the **Export Data** module to your experiment in Studio (classic). You can find this module in the **Input and Output** category.
2. Connect **Export Data** to the module that contain the data you want to export.
3. Double-click **Export Data** to open the **Properties** pane.
4. For **Data destination**, select the type of cloud storage where you'll save your data. If you make any changes to this option, all other properties are reset. So be sure to choose this option first!
5. Provide an account name and authentication method required to access the specified storage account.

Depending on the storage type and whether the account is secured, you might need to provide the account name, file type, access key, or container name. For sources that do not require authentication, generally it is sufficient to know the URL.

For examples of each type, see the following topics:

- [Export to Hive Query](#)
- [Export to Azure SQL Database](#)
- [Export to Azure Table](#)
- [Export to Azure Blob Storage](#)

6. The option, **Use cached results**, lets you repeat the experiment without rewriting the same results each time.

If you deselect this option, results are written to storage each time the experiment is run, regardless of whether the output data has changed.

If you select this option, **Export Data** uses cached data, if available. New results are generated only when there is an upstream change that would affect the results.

7. Run the experiment.

Examples

For examples of how to use the **Export Data** module, see the [Azure AI Gallery](#):

- [Text Classification](#): This sample uses **Export Data** to save intermediate results and then uses **Import Data** to get them from storage for later steps in the experiment.
- [Retail Forecasting Step 1 of 6 - data preprocessing](#): The retail forecasting template illustrates a machine learning task based on data stored in Azure SQL Database. It demonstrates several useful techniques such as how to create an Azure SQL database for machine learning, using the Azure SQL database to pass datasets between experiments in different accounts, saving and combining forecasts.
- [Build and deploy a machine learning model using SQL Server on an Azure VM](#): This article demonstrates how you can use a SQL Server database hosted in an Azure VM as a source for storing training data and the predictions generated by the experiment. It also illustrates how relational database can be used for

feature engineering and feature selection.

- [How to use Azure ML with Azure SQL Data Warehouse](#): This article shows how you can create a machine learning model using data in Azure SQL Data Warehouse.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

TIP

Not sure how or where you should store your data? See this guide to common data scenarios in the data science process: [Scenarios for advanced analytics in Azure Machine Learning](#)

Implementation details

- This module was previously named **Writer**. If you have an existing experiment that uses the **Writer** module, the module is renamed to [Export Data](#) when you refresh the experiment.
- Not all modules produce output that is compatible with [Export Data](#) destinations. For example, [Export Data](#) cannot save a dataset that has been converted to the SVMLight format. [Export Data](#) supports these formats:
 - Dataset (Azure ML internal format)
 - .NET DataTable
 - CSV with or without headers
 - TSV with or without headers

Known issues

- When you select Azure Table as the location to output your data, occasionally there might be an error when writing to the specified table. When this happens, the data might be written to a blob instead.

If this error happens and later you are unable to read from the expected table, try using an Azure storage utility to check the blobs in the specified container in your storage account.

- Currently, you cannot save a blob into a specified Hive table. If you need to write intermediate results, avoid using a Hive table in HDInsight, and use blob storage or table storage instead.
- Currently, if you select HDFS as the location to save output data, this error message is returned: "Microsoft.Analytics.Exceptions.ErrorMapping+ModuleException."

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	The dataset to be written.

Module parameters

This table lists parameters that apply to all [Export Data](#) options. Other parameters are dynamic and change depending on the data destination you select.

Name	Range	Type	Default	Description
Please specify data destination	List	DataSourceOrSink	Blob service in Azure Storage	Indicate whether the data destination is a file in the Blob service , a file in the Table service , a SQL database in Azure, or a Hive table .
Use cached results	TRUE/FALSE	Boolean	FALSE	Select this option to avoid rewriting results unnecessarily. If anything changes upstream in the experiment, Export Data will always execute and write new results. However if nothing has changed, and you have selected this option, Export Data will not execute in order to avoid rewriting the same results.

Exceptions

Exception	Description
Error 0057	An exception occurs when attempting to create a file or blob that already exists.
Error 0001	An exception occurs if one or more specified columns of the dataset couldn't be found.
Error 0027	An exception occurs when two objects have to be of the same size, but they are not.
Error 0079	An exception occurs if the container name in Azure Storage is specified incorrectly.
Error 0052	An exception occurs if the storage access key for the Azure account is specified incorrectly.
Error 0064	An exception occurs if account name or storage access key for the Azure account is specified incorrectly.
Error 0071	An exception occurs if the provided credentials are incorrect.
Error 0018	An exception occurs if the input dataset is not valid.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0003	An exception occurs if one or more inputs are null or empty.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Import Data](#)

[Data Input and Output](#)

[Data Transformation](#)

[Comparing Azure Table Storage and Azure SQL Database](#)

[A-Z Module List](#)

Export to Hive Query

3/10/2021 • 5 minutes to read • [Edit Online](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

This article describes how to use the **Export data to Hive** option in the **Export Data** module in Azure Machine Learning Studio (classic). This option is useful when you are working with very large datasets, and want to save your machine learning experiment data to a Hadoop cluster or HDInsight distributed storage. You might also want to export intermediate results or other data to Hadoop so that you can process it using a MapReduce job.

How to export data to Hive

1. Add the **Export Data** module to your experiment. You can find this module in the **Data Input and Output** category in Azure Machine Learning Studio (classic).

Connect the module to the dataset you want to export.

2. For **Data source**, select **Hive Query**.
3. For **Hive table name** type the name of the Hive table in which to store the dataset.
4. In the **HCatalog server URI** text box, type the fully qualified name of your cluster.

For example, if you created a cluster with the name `mycluster001`, use this format:

`https://mycluster001.azurehdinsight.net`

5. In the **Hadoop user account name** text box, paste in the Hadoop user account that you used when you provisioned the cluster.
6. In the **Hadoop user account password** text box, type the credentials that you used when you provisioned the cluster.
7. For **Location of output data**, select the option that indicates where the data should be stored: HDFS, or Azure.

If the data is in the Hadoop distributed file system (HDFS), it must be accessible via the same account and password that you just entered.

If the data is in Azure, provide the location and credentials of the storage account.

8. If you selected the **HDFS** option, for **HDFS server URI**, specify the HDInsight cluster name without the `https://` prefix.
9. If you selected the **Azure** option, provide the storage account name, and the credentials the module can use to connect to storage.
 - **Azure storage account name:** Type the name of the Azure account. For example, if the full URL of the storage account is `https://myshared.blob.core.windows.net`, you would type `myshared`.

- **Azure storage key:** Copy and paste the key that is provided for accessing the storage account.
 - **Azure container name:** Specify the **default container** for the cluster. For tips on how to figure out the default container, see the [Technical notes](#) section.
10. **Use cached results:** Select this option if you want to avoid rewriting the Hive table each time you run the experiment. If there are no other changes to module parameters, the experiment writes the Hive table only the first time the module is run, or when there are changes to the data.
- If you want to write the Hive table each time the experiment is run, deselect the **Use cached results** option.
11. Run the experiment.

Examples

For examples of how to use the [Export Data](#) module, see the [Azure AI Gallery](#).

- [Advanced Analytics Process and Technology in Action: Using HDInsight Hadoop clusters:](#) This article provides a detailed walkthrough of how to create a cluster, upload data, and call the data from Studio (classic) using Hive.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Common questions

How to avoid out of memory problems when writing large datasets

Sometimes the default configuration of the Hadoop cluster is too limited to support running the MapReduce job. For example, in these [Release Notes](#) for HDInsight, the default settings are defined as a four-node cluster.

If the requirements of the MapReduce job exceed available capacity, the Hive queries might return an **Out of Memory** error message, which causes the [Export Data](#) operation to fail. If this happens, you can change the default memory allocation for Hive queries.

How to avoid re-loading the same data unnecessarily

If you don't want to recreate the Hive table each time you run the experiment, select the **Use cached results** option to TRUE. When this option is set to TRUE, the module will check whether the experiment has run previously, and if a previous run is found, the write operation is not performed.

Usage tips

It can be hard to figure out the default container for the cluster. Here are some tips:

- If you created your cluster by using the default settings, a container with the same name was created at the same time that the cluster was created. That container is the default container for the cluster.
- If you created the cluster by using the **CUSTOM CREATE** option, you were given two options for selecting the default container.

Existing container: If you selected an existing container, that container is the default storage container for the cluster.

Create default container: If you selected this option, a container with the same name as the cluster was created, and you should specify that container name as the default container for the cluster.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Data source	List	Data Source Or Sink	Azure Blob Storage	Data source can be HTTP, FTP, anonymous HTTPS or FTPS, a file in Azure BLOB storage, an Azure table, an Azure SQL Database, a Hive table or an OData endpoint.
Hive table name	any	String	none	Name of table in Hive
HCatalog server URI	any	String	none	Templeton endpoint
Hadoop user account name	any	String	none	Hadoop HDFS/HDIInsight username
Hadoop user account password	any	SecureString	none	Hadoop HDFS/HDIInsight password
Location of output data	any	DataLocation	HDFS	Specify HDFS or Azure for outputDir
HDFS server URI	any	String	none	HDFS rest endpoint
Azure storage account name	any	String	none	Azure storage account name
Azure storage key	any	SecureString	none	Azure storage key
Azure container name	any	String	none	Azure container name
Use cached results	TRUE/FALSE	Boolean	FALSE	Module only executes if valid cache does not exist; otherwise use cached data from prior execution.

Exceptions

EXCEPTION	DESCRIPTION
Error 0027	An exception occurs when two objects have to be the same size, but they are not.
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.

EXCEPTION	DESCRIPTION
Error 0030	an exception occurs in when it is not possible to download a file.
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.
Error 0009	An exception occurs if the Azure storage account name or the container name is specified incorrectly.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Import Data](#)

[Export Data](#)

[Export to Azure SQL Database](#)

[Export to Azure Blob Storage](#)

[Export to Azure Table](#)

Export to Azure SQL Database

3/10/2021 • 6 minutes to read • [Edit Online](#)

This article describes how to use the **Export to Azure SQL Database** option in the [Export Data](#) module in Azure Machine Learning Studio (classic). This option is useful when you want to export data from your machine learning experiment to an Azure SQL Database or Azure SQL Data Warehouse.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Export to a SQL database is useful in many machine learning scenarios: for example, you might want to store intermediate results, save scores, or persist tables of engineered features. Although storing data in an Azure SQL Database or Azure SQL Data Warehouse can be more expensive than using tables or blobs in Azure, there are no transaction fees against SQL databases. Moreover, database storage is ideal for quickly writing smaller amounts of frequently used information, for sharing data between experiments, or for reporting results, predictions, and metrics.

On the other hand, there might be limits on the amount of data that you can store in a database, depending on your subscription type. You should also consider using a database and account that is in the same region as your machine learning workspace.

To export data, you provide the instance name and database name where the data is stored, and run the module using an account that has write permissions. You must also specify the table name, and map the columns from your experiment to columns in the table.

How to export data to an Azure SQL Database

1. Add the [Export Data](#) module to your experiment in Studio (classic). You can find this module in the [Data Input and Output](#) category.
2. Connect **Export data** to the module that produces the data that you want to export.
3. For **Data destination**, select **Azure SQL Database**. This option supports Azure SQL Data Warehouse as well.
4. Indicate the name of the server and database in Azure SQL Database or Azure SQL Data Warehouse.

Database server name: Type the server name as generated by Azure. Typically it has the form

`<generated_identifier>.database.windows.net`

Database name: Type the name of an existing database on the server you just specified. The **Export Data** module cannot create a database.

Server user account name: Type the user name for an account that has access permissions for the database.

Server user account password: Provide the password for the specified user account.

5. Specify the columns to export, and if you want to rename the columns.

Comma-separated list of columns to be saved: Type the names of the columns from the experiment that you want to write to the database.

Data table name: Type the name of the table to store the data in.

For Azure SQL Database, if the table does not exist, a new table is created.

For Azure SQL Data Warehouse, the table must already exist and have the correct schema, so be sure to create it in advance.

Comma-separated list of datatable columns: Type the names of the columns as you wish them to appear in the destination table.

For Azure SQL Database, you can change the column names, but you must keep the columns in the same order that you listed the columns for export, in **Comma-separated list of columns to be saved**.

For Azure SQL Data Warehouse, the columns names must match those already in the destination table schema.

6. **Number of rows written per SQL Azure operation:** This option specifies how many rows should be written to the destination table in each batch.

By default, the value is set to 50, which is the default batch size for Azure SQL Database. However, you should increase this value if you have a large number of rows to write.

For Azure SQL Data Warehouse, we recommend that you set this value to 1. If you use a larger batch size, the size of the command string that is sent to Azure SQL Data Warehouse can exceed the allowed string length, causing an error.

7. **Use cached results:** Select this option to avoid writing new results each time the experiment is run. If there are no other changes to module parameters, the experiment writes the data only the first time the module is run. However, a new write is always performed if any parameters have been changed in [Export Data](#) that would change the results.

8. Run the experiment.

Examples

For examples of how to use the [Export Data](#) module, see the [Azure AI Gallery](#):

- [Retail Forecasting Step 1 of 6 - data-preprocessing](#): The retail forecasting template illustrates a machine learning task based on data stored in Azure SQLDB. It demonstrates useful techniques, such as using Azure SQL database to pass datasets between experiments in different accounts, saving and combining forecasts, and how to create an Azure SQL database just for machine learning.
- [Build and deploy a machine learning model using SQL Server on an Azure VM](#): This article walks you through using a SQL Server database hosted in an Azure VM as a source for storing training data and predictions. It also illustrates how relational database can be used for feature engineering and feature selection.
- [How to use Azure ML with Azure SQL Data Warehouse](#): Demonstrates the use of data from Azure SQL Data Warehouse to build a clustering model.
- [Use Azure Machine Learning with SQL Data Warehouse](#): Demonstrates how to create a regression model to predict pricing, using data in Azure SQL Data Warehouse.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Common questions

Using a database in a different geographical region

If the Azure SQL Database or SQL Data Warehouse is in a different region from the machine learning account, writes might be slower.

Also, you are charged for data ingress and egress on the subscription if the compute node is in a different region than the storage account.

Why are some characters in the output data not displayed correctly

Azure Machine Learning supports the UTF-8 encoding. If string columns in your database use a different encoding, the characters might not be saved correctly.

Also, Azure Machine Learning cannot output data types such as `money`.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Data source	List	Data Source Or Sink	Azure Blob Storage	Data source can be HTTP, FTP, anonymous HTTPS or FTPS, a file in Azure BLOB storage, an Azure table, an Azure SQL Database or Azure SQL Data Warehouse, a Hive table or an OData endpoint.
Database server name	any	String	none	
Database name	any	String	none	
Server user account name	any	String	none	
Server user account password			none	
Comma separated list of columns to be saved			none	
Data table name	any	String	none	
Comma separated list of datatable columns	String	String	none	String
Number of rows written per SQL Azure operation	String	Integer	50	String

Name	Range	Type	Default	Description
Use cached results	TRUE/FALSE	Boolean	FALSE	Module only executes if valid cache does not exist; otherwise use cached data from prior execution.

Exceptions

Exception	Description
Error 0027	An exception occurs when two objects have to be the same size, but they are not.
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	An exception occurs in when it is not possible to download a file.
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.
Error 0009	An exception occurs if the Azure storage account name or the container name is specified incorrectly.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0015	An exception occurs if the database connection has failed.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

- [Import Data](#)
- [Export Data](#)
- [Export to Azure Blob Storage](#)
- [Export to Hive Query](#)
- [Export to Azure Table](#)

Export to Azure Table

3/10/2021 • 7 minutes to read • [Edit Online](#)

This article describes how to use the **Export to Azure** option in the [Export Data](#) module in Azure Machine Learning Studio (classic).

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

This option is useful when you want to export results or intermediate data from a machine learning experiment to an Azure table. The Azure table service is a data management service in Azure that can store large amounts of structured, non-relational data. It is a NoSQL data store that accepts authenticated calls from inside and outside Azure.

How to export data to an Azure table

1. Add the [Export Data](#) module to your experiment. You can find this module in the [Data Input and Output](#) category in Studio (classic).
2. Connect it to the module that produces the data that you want to export to Azure table storage.
3. Specify whether you want to export data to a public shared resource or to a private storage account that requires login credentials, by setting the **Authentication type** option.
 - **Public (SAS URL):** Choose this option if the account supports access via *SAS URL*. In the **Table SAS URI** field, type or paste the full URI that defines the account and the public blob.

The SAS URL is a time bound access URL that you can generate by using an Azure storage utility. In a page accessible via SAS URL, data can be stored using only these formats: CSV, TSV, and ARFF.

 - **Account:** Choose this option if your data is in a **private** account. You must also supply credentials including the account name and the key.
4. If you want to export your data to secured, private storage, provide the credentials needed for accessing the account:
 - **Table account name:** Type or paste the name of the account that contains the blob you want to access. For example, if the full URL of the storage account is `https://myshared.table.core.windows.net`, you would type `myshared`.
 - **Table account key:** Paste the access key that is associated with the storage account.
 - **Table name:** Type the name of the specific table that you want to read.
5. Specify which columns to save to the table store, and which columns to use in defining the table schema, by using the column properties.
 - **Partition key:** Choose the column that should be used for partitioning the saved dataset for the table in Azure Storage. Tables in Azure are partitioned to support load balancing across storage nodes. All table entities are organized by partition; therefore, the **PartitionKey** property is

required for all table operations.

- **Azure table row key:** Choose the column that should be used for the **RowKey** property. The **RowKey** property is a system property that is required for every entity in a table. Along with the **PartitionKey** property, it forms a unique index for every row in the table.

NOTE

You must use different columns for **RowKey** and **PartitionKey**. Make sure that any column you select for the **RowKey** or **PartitionKey** is also included in the list of destination columns, or an error is raised.

- **Azure table origin columns:** Select any additional columns from the dataset that you want to save to the Azure table. You must also include the columns selected for **PartitionKey** and **RowKey**.

For more information about tables in Azure Storage, see [Understanding the Table Service Data Model](#).

6. Specify the names of the columns to write to the table.

IMPORTANT

You must provide a column name for every column that you output to the table, including the **RowKey**, **PartitionKey**, and all origin columns.

If the number of column names you provide does not match the number of output columns, an error is raised.

If you type new column names, they must be provided in the order of the column indexes of the source columns.

7. Azure table write mode:

Indicate how you want the [Export Data](#) to behave when data already exists in the Azure table.

- **Insert:** The `Insert Entity` operation inserts a new entity with a unique primary key, which is formed from a combination of the **PartitionKey** and the **RowKey** properties.
- **Merge:** The `Merge Entity` operation updates an existing entity by updating the entity's properties. This operation does not replace the existing entity.
- **Replace:** The `Update Entity` operation replaces the contents of the given entity in a table.
- **InsertOrReplace:** The `InsertOrReplace Entity` operation inserts the entity if the entity does not exist. If the entity exists, it replaces the existing one.
- **InsertOrMerge:** The `InsertOrMerge Entity` operation inserts the entity if the entity does not exist. If the entity exists, it merges the provided entity properties with the already existing ones.

8. Use cached results:

Indicate whether you want the data to be refreshed each time the experiment is run.

If you select this option, the [Export Data](#) module saves data to the specified table the first time the experiment is run, and thereafter not perform writes, unless there are upstream changes.

If you deselect this option, the data is written to the destination each time the experiment is run, regardless of whether the data is the same or not.

9. Run the experiment.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Common questions

Why did I get an error when writing to an existing Table

Check the schema of the table to make sure the columns names and data types are the same. For example, in Azure table storage, the ID column is expected to be a string.

If you get the error, *Error 0027: The size of passed objects is inconsistent*, verify that the table exists in the specified container. Currently Azure ML can write only to existing tables.

Why do I get the error that an existing column cannot be found

If you have not run the experiment, upstream columns are sometimes not detected by the [Export Data](#). If you make any upstream changes in the experiment, you might need to remove the [Export Data](#) module and then add and reconfigure it.

How can I avoid re-writing the same data unnecessarily

If the data in your experiment changes for any reason, the [Export Data](#) module will always write the new data.

However, if you are running the experiment with other change that do not affect results, set the **Use cached results** option to TRUE. The module will check whether the experiment has run previously using the same options, and if a previous result is found, the data will not be written to the Azure table.

Can I export data to a different geographical region

Yes. However, if the storage account is in a different region from the compute node used for the machine learning experiment, data access might be slower. Further, you are charged for data ingress and egress on the subscription.

Examples

For examples of how to use these machine learning modules, see the [Azure AI Gallery](#).

Module parameters

Public or SAS - Public options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Table SAS URI	any	String		

Account - Private account options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Table account name				
Table account key	any	SecureString		

Storage options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Table name		String	none	

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Partition key	any	SecureString	none	Choose the column to use as the key when partitioning the table. If no column is selected, the column name as the partition key for all entries
Azure table row key	any	ColumnPicker	none	Choose the column that contains the unique identifier for table rows. Defaults to a GUID based row key
Azure table origin columns	any	ColumnPicker	none	Specify which columns to include in the table, either by name or by column index
Azure table destination columns	any	String	none	Type the names of the columns to use in the destination table
Azure table write mode	List: Insert, Merge, Replace, InsertOrReplace, InsertOrMerge	Enum	none	
Use cached results	TRUE/FALSE	Boolean	FALSE	Module only executes if valid cache does not exist; otherwise use cached data from prior execution.

Exceptions

EXCEPTION	DESCRIPTION
Error 0027	An exception occurs when two objects have to be the same size, but they are not.
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	an exception occurs in when it is not possible to download a file.

EXCEPTION	DESCRIPTION
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.
Error 0009	An exception occurs if the Azure storage account name or the container name is specified incorrectly.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

- [Import Data](#)
- [Export Data](#)
- [Export to Azure SQL Database](#)
- [Export to Azure Blob Storage](#)
- [Export to Hive Query](#)

Export to Azure Blob Storage

3/10/2021 • 8 minutes to read • [Edit Online](#)

This article describes how to use the **Export to Azure Blob Storage** option, in the [Export Data](#) module in Azure Machine Learning Studio (classic).

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

This option is useful when you want to export data from a machine learning experiment to Azure blob storage. For example, you might want to share machine learning data outputs with other applications, or store intermediate data or cleaned datasets for use in other experiments.

Azure blobs can be accessed from anywhere, by using either HTTP or HTTPS. Because Azure blob storage is an unstructured data store, you can export data in various formats. Currently, CSV, TSV, and ARFF formats are supported.

To export data to Azure blob for use by other applications, you use the [Export Data](#) module to save the data to Azure blob storage. Then, use any tool that can read data from Azure storage (such as Excel, cloud storage utilities, or other cloud services), to load and use the data.

NOTE

The [Import Data](#) and [Export Data](#) modules can read and write data only from Azure storage created using the Classic deployment model. In other words, the new Azure Blob Storage account type that offers a hot and cool storage access tiers is not yet supported.

Generally, any Azure storage accounts that you might have created before this service option became available should not be affected.

However, if you need to create a new account for use with Azure Machine Learning, we recommend that you either select **Classic** for the **Deployment model**, or use **Resource manager** and for **Account kind**, select **General purpose** rather than **Blob storage**.

How to export data to Azure blob storage

The Azure blob service is for storing large amounts of data, including binary data. There are two types of blob storage: public blobs, and blobs that require login credentials.

1. Add the [Export Data](#) module to your experiment. You can find this module in the [Data Input and Output](#) category in Studio (classic).
2. Connect **Export Data** to the module that produces the data that you want to export to Azure blob storage.
3. Open the **Properties** pane of **Export Data**. For the data destination, select **Azure Blob Storage**.
4. For **Authentication type**, choose **Public (SAS URL)** if you know that the storage supports access via a SAS URL.

A SAS URL is a special type of URL that can be generated by using an Azure storage utility, and is available for only a limited time. It contains all the information that is needed for authentication and download.

For **URI**, type or paste the full URI that defines the account and the public blob.

5. For private accounts, choose **Account**, and provide the account name and the account key, so that the experiment can write to the storage account.

- **Account name:** Type or paste the name of the account where you want to save the data. For example, if the full URL of the storage account is `https://myshared.blob.core.windows.net`, you would type `myshared`.
- **Account key:** Paste the storage access key that is associated with the account.

6. **Path to container, directory, or blob:** Type the name of the blob where the exported data will be stored. For example, to save the results of your experiment to a new blob named **results01.csv** in the container **predictions** in an account named **mymldata**, the full URL for the blob would be `https://mymldata.blob.core.windows.net/predictions/results01.csv`.

Therefore, in the field **Path to container, directory, or blob**, you would specify the container and blob name as follows: `predictions/results01.csv`

7. If you specify the name of a blob that does not already exist, Azure creates the blob for you.

When writing to an existing blob, you can specify that current contents of the blob be overwritten by setting the property, **Azure blob storage write mode**. By default, this property is set to **Error**, meaning that an error is raised whenever an existing blob file of the same name is found.

8. For **File format for blob file**, select the format in which data should be stored.

- **CSV:** Comma-separated values (CSV) is the default storage format. To export column headings together with the data, select the option, **Write blob header row**. For more information about the comma-delimited format used in Azure Machine Learning, see [Convert to CSV](#).
- **TSV:** Tab-separated values (TSV) format is compatible with many machine learning tools. To export column headings together with the data, select the option, **Write blob header row**. For more information about the tab-separated format used in Azure Machine Learning, see [Convert to TSV](#).
- **ARFF:** This format supports saving files in the format used by the Weka toolset. This format is not supported for files stored in a SAS URL. For more information about the ARFF format, see [Convert to ARFF](#).

9. **Use cached results:** Select this option if you want to avoid rewriting the results to the blob file each time you run the experiment. If there are no other changes to module parameters, the experiment writes the results only the first time the module is run, or when there are changes to the data.

Examples

For examples of how to use the [Export Data](#) module, see the [Azure AI Gallery](#):

- [Convert Dataset to VW Format](#): This experiment uses Python script together with the [Export Data](#) module to create data that can be used by Vowpal Wabbit.
- [Setting up predictive analytics pipelines using Azure SQL Data Warehouse](#): This scenario describes data movement among multiple components, including Azure Machine Learning and SQL Data Warehouse.
- [No-code batch scoring](#): This tutorial demonstrates how you can use Azure Logic Apps to automate both the import of data used by experiments, and writing experiment results to blob storage.

- [Operationalize Azure ML solution with On-premise SQL Server using Azure data factory](#): This article describes a more complex data pipeline that sends data back to an on-premises SQL Server database, using blob storage as an interim stage. Use of an on-premises database requires configuration of a data gateway, but you can skip that part of the example, and just use blob storage.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Common questions

How can I avoid writing the data if the experiment hasn't changed?

When your experiment results changes, [Export Data](#) always saves the new dataset. However, if you are running the experiment repeatedly without making changes that affect the output data, you can select the **Use cached results** option.

The module checks whether the experiment has run previously using the same data and the same options, and if a previous run is found, the write operation is not repeated.

Can I save data to an account in a different geographical region?

Yes, you can write data to accounts in different regions. However, if the storage account is in a different region from the compute node used for the machine learning experiment, data access might be slower. Also, you are charged for data ingress and egress on the subscription.

Module parameters

General options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Data source	List	Data Source Or Sink	Azure Blob Storage	The destination can be a file in Azure BLOB storage, an Azure table, a table or view in an Azure SQL Database, or a Hive table.
Use cached results	TRUE/FALSE	Boolean	FALSE	Module only executes if valid cache does not exist; otherwise use cached data from prior execution.
Please specify authentication type	SAS/Account	AuthenticationType	Account	Indicates whether SAS or account credentials should be used for access authorization

Public or SAS - Public storage options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
SAS URI for blob	any	String	none	The SAS URI of the blob to be written to (required)

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
File format for SAS file	ARFF	LoaderUtils.FileTypes	CSV	Indicates whether file is CSV, TSV, or ARFF. (required)
	CSV			
	TSV			
Write SAS header row	TRUE/FALSE	Boolean	FALSE	Indicates whether column headings should be written to the file

Account - Private storage options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Azure account name	any	String	none	Azure user account name
Azure account key	any	SecureString	none	Azure storage key
Path to blob beginning with container	any	String	none	Name of the blob file, beginning with the container name
Azure blob storage write mode	List: Error, Overwrite	enum:BlobFileWriteMode	Error	Choose the method of writing blob files
File format for blob file	ARFF	LoaderUtils.FileTypes	CSV	Indicates whether blob file is CSV, TSV, or ARFF
	CSV			
	TSV			
Write blob header row	TRUE/FALSE	Boolean	FALSE	Indicates whether blob file should have header row

Exceptions

EXCEPTION	DESCRIPTION
Error 0027	An exception occurs when two objects have to be the same size, but they are not.
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	an exception occurs in when it is not possible to download a file.

EXCEPTION	DESCRIPTION
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.
Error 0009	An exception occurs if the Azure storage account name or the container name is specified incorrectly.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Import Data](#)

[Export Data](#)

[Export to Azure SQL Database](#)

[Export to Hive Query](#)

[Export to Azure Table](#)

Import Data

3/10/2021 • 11 minutes to read • [Edit Online](#)

Loads data from external sources on the web; from various forms of cloud-based storage in Azure such as tables, blobs, and SQL databases; and from on-premises SQL Server databases

Category: [Data Input and Output](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Import Data](#) module in Azure Machine Learning Studio (classic), to load data into a machine learning experiment from existing cloud data services.

The module now features a wizard to help you choose a storage option and select from among existing subscriptions and accounts to quickly configure all options. Need to edit an existing data connection? No problem; the wizard loads all previous configuration details so that you don't have to start again from scratch.

After you define the data you want and connect to the source, [Import Data](#) infers the data type of each column based on the values it contains, and loads the data into your Azure Machine Learning Studio (classic) workspace. The output of [Import Data](#) is a dataset that can be used with any experiment.

IMPORTANT

Currently, there are limitations on the types of storage accounts that are supported. For more information, see [Technical Notes](#).

If your source data changes, you can refresh the dataset and add new data by re-running [Import Data](#). However, if you don't want to re-read from the source each time you run the experiment, select the **Use cached results** option to TRUE. When this option is selected, the module checks whether the experiment has run previously using the same source and same input options. If a previous run is found, the data in the cache is used, instead of re-loading the data from the source.

NOTE

This module was previously named **Reader**. If you previously used the **Reader** module in an experiment, it is renamed to [Import Data](#) when you refresh the experiment.

Data sources

The [Import Data](#) module supports the following data sources. Click the links for detailed instructions and examples of using each data source.

If you are not sure how or where you should store your data, see this guide to common data scenarios in the data science process: [Scenarios for advanced analytics in Azure Machine Learning](#).

DATA SOURCE	USE WITH
Web URL via HTTP	Get data that is hosted on a web URL that uses HTTP and that has been provided in the CSV, TSV, ARFF, or SvmLight formats
Hive Query	Get data from distributed storage in Hadoop. You specify the data you want by using the HiveQL language
Azure SQL Database	Get data from Azure SQL Database or from Azure SQL Data Warehouse
Azure Table	Get data that is stored in the Azure table service
Import from Azure Blob Storage	Get data that is stored in the Azure blob service
Data Feed Providers	Get data exposed as a feed in OData format
Import from On-Premises SQL Server Database	Get data from an on-premises SQL Server database using the Microsoft Data Management Gateway
Azure Cosmos DB	Get data stored in JSON format in Azure Cosmos DB.

TIP

Need to import data in the JSON format? Both R and Python support REST APIs, so use the [Execute Python Script](#) or [Execute R Script](#) modules to parse your data and save it as an Azure ML dataset.

Or, use the SQL DB API for CosmosDB, which supports multiple JSON stores, including MongoDB, to read your data using the [Import from Azure Cosmos DB](#) option. For more information, see [Import from Azure Cosmos DB](#).

How to use Import Data

1. Add the **Import Data** module to your experiment. You can find this module in the **Data Input and Output** category in Studio (classic).

2. Click **Launch Data Import Wizard** to configure the data source using a wizard.

The wizard gets the account name and credentials, and help you configure other options. If you are editing an existing configuration, it loads the current values first.

3. If you do not want to use the wizard, click **Data source**, and choose the type of cloud-based storage you are reading from.

Additional settings depend on the type of storage you choose, and whether the storage is secured or not. You might need to provide the account name, file type, or credentials. Some sources do not require authentication; for others, you might need to know the account name, a key, or container name.

For details, see the list of [Data sources](#).

4. Select the **Use cached results** option if you want to cache the dataset for re-use on successive runs.

Assuming there have been no other changes to module parameters, the experiment loads the data only the first time the module is run, and thereafter uses a cached version of the dataset.

Deselect this option if you need to reload the data each time you run the experiment.

5. Run the experiment.

When [Import Data](#) loads the data into Studio (classic), it infers the data type of each column based on the values it contains, either numerical or categorical.

- If a header is present, the header is used to name the columns of the output dataset.
- If there are no existing column headers in the data, new column names are generated using the format col1, col2,... ,coln.

Results

When import completes, click the output dataset and select [Visualize](#) to see if the data was imported successfully.

If you want to save the data for re-use, rather than importing a new set of data each time the experiment is run, right-click the output and select [Save as Dataset](#). Choose a name for the dataset. The saved dataset preserves the data at the time of saving, and data is not updated when the experiment is re-run, even if the dataset in the experiment changes. This can be handy for taking snapshots of data.

After importing the data, it might need some additional preparations for modeling and analysis:

- Generate statistical summaries of the data, using [Summarize Data](#) or [Compute Elementary Statistics](#).
- Use [Edit Metadata](#) to change column names, to handle a column as a different data type, or to indicate that some columns are labels or features.
- Use [Select Columns in Dataset](#) to select a subset of columns to transform or use in modeling. The transformed or removed columns can easily be rejoined to the original dataset by using the [Add Columns](#) module or the [Join Data](#) module.
- Use [Partition and Sample](#) to divide the dataset, perform sampling, or get the top n rows.
- Use [Apply SQL Transformation](#) to aggregate data, filter, or transform using SQL statements.
- Use these modules to clean up text columns and generate new text features:
 - [Preprocess Text](#)
 - [Extract N-Gram Features from Text](#)
 - [Named Entity Recognition](#)
 - [Execute Python Script](#), to implement custom NLP based on [nltk](#).

Technical notes

This section provides a list of known issues with the [Import Data](#) module, as well as some general troubleshooting information not specific to a source type.

Supported account types

Frequently Azure releases new services or new storage types; however, there is typically a delay while support for new account types is implemented in Azure Machine Learning Studio (classic).

- Currently, Azure Machine Learning supports all general purpose storage accounts, except for those using zone-redundant storage (ZRS).
- Locally redundant storage (LRS) and geo-redundant storage options are supported.
- Block blobs are supported but Append blobs are not.

Common questions and issues

This section describes some known issues, common questions, and workarounds.

Headers must be single rows

If you are importing from CSV files, be aware that Azure Machine Learning allows a single header row. You cannot insert multi-line headers.

Custom separators supported on import but not export

The **Import Data** module supports importing data that uses alternative column separators, such as the semicolon (;), which is often used in Europe. When you import data from CSV files in external storage, select the **CSV with encodings** option, and pick a supported encoding.

However, you cannot generate alternative separators when you prepare data for export using the [Convert to CSV](#) module.

Poor column separation on string data containing commas

Just about every character that can be specified as a column separator (tabs, spaces, commas, etc.) can also be found randomly in text fields. Importing text from CSV always requires caution to avoid separating text across unnecessary new columns. It is a common problem in text processing that you have probably encountered and handled in different ways.

Problems can also occur when you try to export a column of string data that contains commas. Azure Machine Learning does not support any special handling or special translation of such data, such as enclosing strings in quotation marks. Also, you cannot use escape characters before a comma to ensure that commas are handled as a literal character. AS a consequence, new fields are created in the output file for each comma that is encountered in the string field.

To avoid problems on export, use the [Preprocess Text](#) module to remove punctuation characters from string fields.

You can also use custom [R script](#) or [Python script](#) to process complex text and ensure that data can be imported or exported correctly.

UTF-8 encoding required

Azure Machine Learning requires UTF-8 encoding. If the data you are importing uses a different encoding, or was exported from a data source that uses a different default encoding, various problems might appear in the text.

For example, the following image contains the same multilanguage dataset exported from Excel and then imported into Azure Machine Learning under four different combinations of file type and encoding.

Import from xls file with default encoding

```
##0.00_); ##0.00  
[Red] \\"$"\#
```

Import from xlsx file with default encoding

```
    (%  
   "D '4j 0u2js    
MY    S         
f   C    y   I<  
   f       
```

Import from .txt file with CSV format but default encoding

E ' stato un bellissimo hotel con un personale cordiale e buon servizio	Italian
???????????????????????????	Japanese
????????	

C�est un magnifique h�tel avec un personnel sympathique et un service de qualit�	French
Va ser un magn�fic hotel amb un personal amable i bon servei	Catalan

Import from CSV file with UTF-8 encoding

E ' stato un bellissimo hotel con un personale cordiale e buon servizio	Italian
素晴らしいホテルで、スタッフはフレンドリーで、サービスも良かつた	Japanese

The third example represents data that was lost during while saving from Excel in CSV format, because the correct encoding was not specified at that time. Therefore, if you run into problems, be sure to check not just the file you are importing from, but whether the file was correctly exported from the source.

Dataset does not have column names

If the dataset you are importing does not have column names, be sure to specify one of the "no header" options. When you do so, Import Data adds default column names using the format Col1, Col2, etc. Later, use [Edit Metadata](#) to fix the column names.

If you are exporting a dataset to a CSV file, use [Edit Metadata](#) to add column names before converting or exporting it.

Workarounds for unsupported data sources

If you need to get data from a source that is not in the list, there are various workarounds you can try:

- To upload data from a file on your computer, click **New** in Studio (classic), select **Dataset**, and then select **From Local File**. Locate the file and specify the format (TSV, CSV, etc.). For more information, see [Import training data into Studio \(classic\)](#).
- Use R or Python. You can use the [Execute R Script](#) module with an appropriate R package to get data from other cloud databases.

The [Execute Python Script](#) module also lets you read and convert data from a variety of sources. See these examples from Microsoft data scientists in the Cortana Intelligence Gallery:

[Convert PDF to Text](#)

[Load non-text file from Azure blob storage](#)

- Get data from AWS clusters. You can run a query against a generic Hive cluster with WebHCat or HCatalog endpoint enabled. Or publish as a page and read from the Web URL.
- Get data from MongoDB. The data migration utility for Azure Cosmos DB supports a wide variety of sources and formats. For more information and examples, see [Azure Cosmos DB: Data migration tool](#)

For more ideas and workarounds, see the [Azure Machine Learning forum](#) or [Azure AI Gallery](#).

Module parameters

Each data source must be configured using different options. This table lists only the options that are common to all data sources.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Data source	List	DataSource Or Sink	Blob service in Azure Storage	Data source can be HTTP, anonymous HTTPS, a file in the Blob service or Table service, a SQL database in Azure, an Azure SQL Data Warehouse, a Hive table, or an OData endpoint.
Use cached results	TRUE/FALSE	Boolean	FALSE	If TRUE, the module will check whether the experiment has run previously using the same source and same input options, and if a previous run is found, the data in the cache is used. If FALSE, or if changes are found, data will be reloaded from the source.

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with downloaded data

Exceptions

EXCEPTION	DESCRIPTION
Error 0027	An exception occurs when two objects have to be the same size, but they are not.

EXCEPTION	DESCRIPTION
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	an exception occurs in when it is not possible to download a file.
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.
Error 0009	An exception occurs if the Azure storage account name or the container name is specified incorrectly.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0015	An exception occurs if the database connection has failed.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Data Input and Output](#)

[Data Format Conversions](#)

[Export Data](#)

[A-Z Module List](#)

Import from Web URL via HTTP

3/10/2021 • 6 minutes to read • [Edit Online](#)

This article describes how to use the [Import Data](#) module in Azure Machine Learning Studio (classic), to read data from a public Web page for use in a machine learning experiment.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The following restrictions apply to data published on a web page:

- Data must be in one of the supported formats: CSV, TSV, ARFF, or SvmLight. Other data will cause errors.
- No authentication is required or supported. Data must be publicly available.

How to import data via HTTP

There are two ways to get data: use the wizard to set up the data source, or configure it manually.

Use the Data Import Wizard

1. Add the [Import Data](#) module to your experiment. You can find the module in Studio (classic), in the **Data Input and Output** category.
2. Click **Launch Import Data Wizard** and select **Web URL via HTTP**.
3. Paste in the URL, and select a data format.
4. When configuration is complete, right-click the module, and select **Run Selected**.

To edit an existing data connection, start the wizard again. The wizard loads all previous configuration details so that you don't have to start again from scratch.

Manually set properties in the Import Data module

The following steps describe how to manually configure the import source.

1. Add the [Import Data](#) module to your experiment. You can find the module in Studio (classic), in the **Data Input and Output** category.
2. For **Data source**, select **Web URL via HTTP**.
3. For **URL**, type or paste the full URL of the page that contains the data you want to load.

The URL should include the site URL and the full path, with file name and extension, to the page that contains the data to load.

For example, the following page contains the Iris data set from the machine learning repository of the University of California, Irvine:

`https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data`

4. For **Data format**, select one of the supported data formats from the list.

We recommend that you always check the data beforehand to determine the format. The UC Irvine page uses the CSV format. Other supported data formats are TSV, ARFF, and SvmLight.

5. If the data is in CSV or TSV format, use the **File has header row** option to indicate whether or not the source data includes a header row. The header row is used to assign column names.
6. Select the **Use cached results** options if you don't expect the data to change much, or if you want to avoid reloading the data each time you run the experiment.

When this option is selected, the experiment loads the data the first time the module is run, and thereafter uses a cached version of the dataset.

If you want to re-load the dataset on each iteration of the experiment dataset, deselect the **Use cached results** option. Results are also re-loaded if there are any changes to the parameters of [Import Data](#).

7. Run the experiment.

Results

When complete, click the output dataset and select **Visualize** to see if the data was imported successfully.

Examples

See these examples in the [Azure AI Gallery](#) of machine learning experiments that get data from public web sites:

- [Letter Recognition sample](#): Gets a training dataset from the public machine learning repository hosted by UC Irvine.
- [Download UCI Dataset](#): Reads a dataset in the CSV format.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Common questions

Can I filter data as it is being read from the source

No. That option is not supported with this data source.

After reading the data into Azure Machine Learning Studio (classic), you can split the dataset, use sampling, and so forth to get just the rows you want:

- Write some simple R code in the [Execute R Script](#) to get a portion of the data by rows or columns.
- Use the [Split Data](#) module with a relative expression or a regular expression to isolate the data you want.
- If you loaded more data than you need, overwrite the cached dataset by reading a new dataset, and saving it with the same name.

How can I avoid re-loading the same data unnecessarily

If your source data changes, you can refresh the dataset and add new data by re-running [Import Data](#).

If you don't want to re-read from the source each time you run the experiment, select the **Use cached results** option to TRUE. When this option is set to TRUE, the module checks whether the experiment has run previously using the same source and same input options. If a previous run is found, the data in the cache is used, instead of re-loading the data from the source.

Why was an extra row added at the end of my dataset

If the [Import Data](#) module encounters a row of data that is followed by an empty line or a trailing new line character, an extra row is added at the end of the table. This new row contains missing values.

The reason for interpreting a trailing new line as a new row is that [Import Data](#) cannot determine the difference

between an actual empty line and an empty line that is created by the user pressing ENTER at the end of a file.

Because some machine learning algorithms support missing data and would thus treat this line as a case (which in turn could affect the results), you should use [Clean Missing Data](#) to check for missing values (particularly rows that are completely empty), and remove them as needed.

Before you check for empty rows, you might also want to divide the dataset by using [Split Data](#). This separates rows with partial missing values, which represent actual missing values in the source data. Use the **Select head N rows** option to read the first part of the dataset into a separate container from the last line.

Why are some characters in my source file not displayed correctly

Azure Machine Learning supports the UTF-8 encoding. If your source file used another type of encoding, the characters might not be imported correctly.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Data source	List	Data Source Or Sink	Azure Blob Storage	Data source can be HTTP, FTP, anonymous HTTPS or FTPS, a file in Azure BLOB storage, an Azure table, an Azure SQL Database, an on-premises SQL Server database, a Hive table, or an OData endpoint.
URL	any	String	none	URL for HTTP
Data format	CSV TSV ARFF SvmLight	Data Format	CSV	File type of HTTP source
CSV or TSV has header row	TRUE/FALSE	Boolean	false	Indicates if CSV or TSV file has a header row
Use cached results	TRUE/FALSE	Boolean	FALSE	Module executes only if valid cache does not exist. Otherwise, cached data from previous execution is used.

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with downloaded data

Exceptions

EXCEPTION	DESCRIPTION
Error 0027	An exception occurs when two objects have to be the same size, but they are not.
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	An exception occurs in when it is not possible to download a file.
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

- [Import Data](#)
- [Export Data](#)
- [Import from Hive Query](#)
- [Import from Azure SQL Database](#)
- [Import from Azure Table](#)
- [Import from Azure Blob Storage](#)
- [Import from Data Feed Providers](#)
- [Import from On-Premises SQL Server Database](#)

Import from Hive Query

3/10/2021 • 9 minutes to read • [Edit Online](#)

This article describes how to use the [Import Data](#) module in Azure Machine Learning Studio (classic), to get data from Hadoop clusters and HDInsight distributed storage.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Importing data from Hive is particularly useful for loading large datasets, or if you want to pre-process the data using a MapReduce job before loading the data into a machine learning experiment.

IMPORTANT

As of July 31, 2018, Microsoft Azure HDInsight version 3.3 was the last version of HDInsight on Windows. If you have any HDInsight clusters on Windows 3.3 or earlier, you must migrate to HDInsight on Linux (HDInsight version 3.5 or later).

Please see the [Retired versions](#) section for more information on retired versions of HDInsight. Azure Machine Learning Studio (classic) will support [HDInsight on Linux](#) in certain scenarios.

Support for HDInsight on Linux

Azure Machine Learning Studio (classic) has support for HDInsight on Linux in the following scenarios:

- Hadoop 2.7.3 (HDI 3.6) Blob as default, ADLS secondary
- Spark 2.1.0 (HDI 3.6) Blob as default, ADLS secondary
- Spark 2.2.0 (HDI 3.6) Blob as default, ADLS secondary
- Spark 2.3.0 (HDI 3.6) Blob as default, ADLS secondary

Known Issues

There are several known issues with using the Import Data module for Hive Queries with HDInsight on Linux:

- Enterprise Security Package is not supported
- [/tmp/hive not writeable](#)

How to import data from Hive queries

Use the wizard

The module features a new wizard to help you choose a storage option, select from among existing subscriptions and accounts, and quickly configure all options.

1. Add the [Import Data](#) module to your experiment. You can find the module in Studio (classic), in the [Data Input and Output](#) category.
2. Click **Launch Import Data Wizard** and follow the prompts.
3. When configuration is complete, to actually copy the data into your experiment, right-click the module, and select **Run Selected**.

If you need to edit an existing data connection, the wizard loads all previous configuration details so that you don't have to start again from scratch.

Manually set import properties

The following steps describe how to manually configure the import source.

1. Add the **Import Data** module to your experiment. You can find the module in Studio (classic), in the **Data Input and Output** category.
2. For **Data source**, select **Hive Query**.
3. In the **Hive database query** text box, specify the data you want to read by using HiveQL.

HiveQL is a SQL-like query language that can also be used to aggregate data and perform data filtering before you add the data to Machine Learning Studio (classic). However, the Hive query must return the data in a tabular format.

For example, this statement is a valid Hive query:

```
SELECT <column list> FROM table WHERE <expression>;
```

4. Click the **HCatalog server URI** text box, and then type the fully qualified name of your cluster.

For example, if you created a cluster with the name mycluster001, use this format:

```
https://mycluster001.azurehdinsight.net
```

5. Click the **Hadoop user account name** text box, and paste in the Hadoop user account that you used when you provisioned the cluster.
6. Click the **Hadoop user account password** text box, and type the credentials that you used when you provisioned the cluster.

For more information about cluster naming and authentication for Hadoop, see [Provision Hadoop clusters in HDInsight](#).

7. For **Location of output data**, select the option that indicates where the data is stored. If the data is in the Hadoop distributed file system (HDFS), it must be accessible via the same account and password that you just entered. If the data is in Azure, provide the location and credentials of the storage account.
 - **HDFS**: Type or paste the HDFS server URI. Be sure to use the HDInsight cluster name **without** the `HTTPS://` prefix.
 - **Azure**: For **Azure storage account name**, type the name of the Azure account. For example, if the full URL of the storage account is `https://myshared.blob.core.windows.net`, you would type `myshared`.
 - **Azure storage key**: Copy and paste the key that is provided for accessing the storage account.
 - For **Azure container name**, specify the **default container** for the cluster. See the [Tips](#) section for help figuring out which container to use.
8. Select the **Use cached results** options if you don't expect the data to change much, or if you want to avoid reloading the data each time you run the experiment.

When this is selected, if there are no other changes to module parameters, the experiment loads the data the first time the module is run, and thereafter uses a cached version of the dataset.

If you want to re-load the dataset on each iteration of the experiment dataset, deselect the **Use cached results** option. Results are also re-loaded when there are changes to the parameters of [Import Data](#).

9. Run the experiment.

Results

When complete, click the output dataset and select **Visualize** to see if the data was imported successfully.

If you get errors, check your data for missing values, additional empty columns, or incompatible data types.

Examples

For examples of how to configure an HDInsight cluster and use Hive queries in machine learning experiments, see these resources:

- This article provides a detailed walkthrough of how to create a cluster, upload data, and call the data from Studio (classic) using Hive: [Advanced Analytics Process and Technology in Action: Using HDInsight Hadoop clusters](#).
- This blog by MVP Vesa Tikkanen describes some issues and workarounds when reading very large files (distributed queries) from an HD cluster on Linux: [Reading Linux HDInsight Hive from Azure ML](#)

Although Hive offers superior features for many kinds of data clean-up and pre-processing, after import, you might find these tools useful for preparing the data for modeling:

- Use the [Edit Metadata](#) and other modules to change column names, specify which columns contain labels and features, and specify the column data type. For examples, see [Dataset Processing](#).
- Post-process text data using Python, to remove punctuation, flag parts of speech, and much more. For examples, see [Text Classification](#).
- Combine multiple tables from different sources into a single table of training data. For examples, see [Predictive maintenance](#).

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

How to determine the default container

If you created your cluster by accepting all defaults, a container with the same name as the cluster was created at the same time the cluster was created. That container is the default container for the cluster. However, if you chose the **CUSTOM CREATE** option when creating a cluster, you are given two options for selecting the default container. The first option is to select an existing container. When you do so, that container becomes the default storage container for the cluster. The second option is **Create default container**. When you use this option, the default container has the same name as the cluster.

How to call Python scripts from a Hive query

You can use the [Import Data](#) module to run Hive queries that call Python UDFs to process records.

For more information, see [Use Python with Hive and Pig in HDInsight](#).

Avoiding out of memory problems when using Hive to pre-process data

When using Hive queries to extract records from big data sources, sometimes the default configuration of the Hadoop cluster is too limited to support running the MapReduce job. For example, in these [Release Notes](#) for HDInsight, the default settings are defined as a four-node cluster.

If the requirements of the MapReduce job exceed available capacity, the Hive queries might return an **Out of Memory** error message, which causes the [Import Data](#) operation to fail. If this happens, you can change the default memory allocation for Hive queries in the [Import Data](#) module, as shown here:

Properties

▲ Reader

Data source

Hive Query

Hive database query

```

1 set mapreduce.map.memory.mb = 2048;
2 set mapreduce.reduce.memory.mb=6144;
3 set mapreduce.reduce.java.opts=-Xmx12000m;
4 set mapred.reduce.tasks=128;
5 set mapred.tasktracker.reduce.tasks.maximum=128;
6

```

In this example, the commands `set mapreduce.map.memory.mb` and `set mapreduce.reduce.memory.mb` are used to increase the amount of memory, to use the maximum allowed in the cluster.

Common questions

How can I avoid re-loading the same data unnecessarily

If your source data changes, you can refresh the dataset and add new data by re-running [Import Data](#). However, if you don't want to re-read from the source each time you run the experiment, select the **Use cached results** option to TRUE. When this option is set to TRUE, the module will check whether the experiment has run previously using the same source and same input options, and if a previous run is found, the data in the cache is used, instead of re-loading the data from the source.

Can I filter data as it is being read from the source

The [Import Data](#) module itself does not support filtering as data is being read.

To filter data before reading it into Azure Machine Learning Studio (classic), use a Hive query or a MapReduce job to aggregate and transform the data.

There are also multiple options for filtering data after it has been loaded into Azure Machine Learning Studio (classic):

- Use a custom R script to get only the data you want.
- Use the [Split Data](#) module with a relative expression or a regular expression to isolate the data you want, and then save it as a dataset.

NOTE

If you find that you have loaded more data than you need, you can overwrite the cached dataset by reading a new dataset, and saving it with the same name as the older, larger data.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
------	-------	------	---------	-------------

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Data source	List	Data Source Or Sink	Azure Blob Storage	Data source can be HTTP, FTP, anonymous HTTPS or FTPS, a file in Azure BLOB storage, an Azure table, an Azure SQL Database, an on-premises SQL Server database, a Hive table, or an OData endpoint.
Hive database query	any	StreamReader		HQL query
HCatalog server URI	any	String		Templeton endpoint
Hadoop user account name	any	String		Hadoop HDFS/HDInsight username
Hadoop user account password	any	SecureString		Hadoop HDFS/HDInsight password
Location of output data	any	DataLocation	HDFS	Specify HDFS or Azure for outputDir
HDFS server URI	any	String		HDFS rest endpoint
Azure storage account name	any	String		Azure storage account name
Azure storage key	any	SecureString		Azure storage key
Azure container name	any	String		Azure container name
Data content type	List (subset)	Url Contents	OData	Data format type
Source URL	any	String		URL for Power Query data source
Use cached results	TRUE/FALSE	Boolean	FALSE	description

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with downloaded data

Exceptions

EXCEPTION	DESCRIPTION
Error 0027	An exception occurs when two objects have to be the same size, but they are not.
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	An exception occurs in when it is not possible to download a file.
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.
Error 0009	An exception occurs if the Azure storage account name or the container name is specified incorrectly.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0015	An exception occurs if the database connection has failed.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

- [Import Data](#)
- [Export Data](#)
- [Import from Web URL via HTTP](#)
- [Import from Azure SQL Database](#)
- [Import from Azure Table](#)
- [Import from Azure Blob Storage](#)
- [Import from Data Feed Providers](#)
- [Import from On-Premises SQL Server Database](#)

Import from Azure SQL Database

3/10/2021 • 7 minutes to read • [Edit Online](#)

This article describes how to use the [Import Data](#) module in Azure Machine Learning Studio (classic), to get data from an Azure SQL Database or Azure SQL Data Warehouse.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

To import data from a database, you must specify both the server name and database name, and a SQL statement that defines the table, view, or query.

In general, storing data in Azure databases is more expensive than using tables or blobs in Azure. There may also be limits on the amount of data that you can store in a database, depending on your subscription type. However, there are no transaction fees against SQL Azure Database, so that option is ideal for fast access to smaller amounts of frequently used information, such as data lookup tables or data dictionaries.

Storing data in an Azure database is also preferred if you need to be able to filter data before reading it, or if you want to save predictions or metrics back to the database for reporting.

How to import data from Azure SQL Database or SQL Data Warehouse

Use the Data Import Wizard

The module features a new wizard to help you choose a storage option, select from among existing subscriptions and accounts, and quickly configure all options.

1. Add the [Import Data](#) module to your experiment. You can find the module in Studio (classic), in the [Data Input and Output](#) category.
2. Click **Launch Import Data Wizard** and follow the prompts.
3. When configuration is complete, to actually copy the data into your experiment, right-click the module, and select **Run Selected**.

If you need to edit an existing data connection, the wizard loads all previous configuration details so that you don't have to start again from scratch.

Manually set properties in the Import Data module

The following steps describe how to manually configure the import source.

1. Add the [Import Data](#) module to your experiment. You can find this module in Studio (classic), in the [Data Input and Output](#) category.
2. For **Data source**, select **Azure SQL Database**.
3. Set the following options specific to Azure SQL Database or Azure SQL Data Warehouse.

Database server name: Type the server name that is generated by Azure. Typically it has the form

```
<generated_identifier>.database.windows.net .
```

Database name: Type the name of an existing database on the server you just specified.

Server user account name: Type the user name of an account that has access permissions for the database.

Server user account password: Provide the password for the specified user account.

Database query: Type or paste a SQL statement that describes the data you want to read. Always validate the SQL statement and verify the query results beforehand, using a tool such as Visual Studio Server Explorer or SQL Server Data Tools.

NOTE

Import Data module only supports inputting Database name, user account name and password as credentials.

4. If the dataset that you read into Azure Machine Learning is not expected to change between runs of the experiment, select the **Use cached results** option.

When this is selected, if there are no other changes to module parameters, the experiment loads the data the first time the module is run, and thereafter uses a cached version of the dataset.

If you want to re-load the dataset on each iteration of the experiment, deselect this option. The dataset is reloaded from the source each time any parameters are changed in [Import Data](#).

5. Run the experiment.

As [Import Data](#) loads the data into Studio (classic), some implicit type conversion might also be performed, depending on the data types used in the source database.

Results

When import is complete, click the output dataset and select **Visualize** to see if the data was imported successfully.

Optionally, you can change the dataset and its metadata using the tools in Studio (classic):

- Use [Edit Metadata](#) to change column names, convert a column to a different data type, or to indicate which columns are labels or features.
- Use [Select Columns in Dataset](#) to select a subset of columns.
- Use [Partition and Sample](#) to separate the dataset by criteria, or get the top n rows.

Examples

For an example of how to use data from Azure databases in machine learning, see these articles and experiments:

- [Retail Forecasting Step 1 of 6 - data preprocessing](#): The Retail forecasting template illustrates a typical scenario that uses data stored in Azure SQLDB for analysis.

It also demonstrates some useful techniques, such as using Azure SQLDB to passing datasets between experiments in different accounts, saving and combining forecasts, and how to create an Azure SQLDB for machine learning.

- [Use Azure Machine Learning with SQL Data Warehouse](#): This article demonstrates how to create a regression model to predict prices using Azure SQL Data Warehouse.

- [How to use Azure ML with Azure SQL Data Warehouse](#): This article builds a clustering model on AdventureWorks, using [Import Data](#) and [Export Data](#) with Azure SQL Data Warehouse.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Common questions

Can I filter data as it is being read from the source?

The [Import Data](#) module does not support filtering as data is being read. We recommend that you create a view or define a query that generates only the rows you need.

NOTE

If you find that you have loaded more data than you need, you can overwrite the cached dataset by reading a new dataset, and saving it with the same name as the older, larger data.

Why do I get the error, "Type Decimal is not supported"?

When reading data from a SQL database, you might encounter an error message reporting an unsupported data type.

If the data you get from the SQL database includes data types that are not supported in Azure Machine Learning, you should cast or convert the decimals to a supported data before reading the data. [Import Data](#) cannot automatically perform any conversions that would result in a loss of precision.

For more information about supported data types, see [Module Data Types](#).

What happens if the database is in a different geographical region. Can Import Data still access the database? Where is the data stored?

If the database is in a different region from the machine learning account, data access might be slower. Further, you are charged for data ingress and egress on the subscription if the compute node is in a different region than the storage account.

Data that you read into your workspace for an experiment is saved in the storage account associated with the experiment.

Why are some characters not displayed correctly?

Azure Machine Learning supports the UTF-8 encoding. If string columns in your database use a different encoding, the characters might not be imported correctly.

One option is to export the data to a CSV file in Azure storage, and use the option **CSV with encoding** to specify parameters for custom delimiters, the code page, and so forth.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
------	-------	------	---------	-------------

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Data source	List	Data Source Or Sink	Azure Blob Storage	Data source can be HTTP, FTP, anonymous HTTPS or FTPS, a file in Azure BLOB storage, an Azure table, an Azure SQL Database, an on-premises SQL Server database, a Hive table, or an OData endpoint.
HDFS server URI	any	String	none	HDFS rest endpoint
Database server name	any	String	none	Azure storage account name
Database name	any	SecureString	none	Azure storage key
Server user account name	any	String	none	Azure container name
Server user account name	List (subset)	Url Contents	OData	Data format type
Database query	any	String	none	Data format type
Use cached results	TRUE/FALSE	Boolean	FALSE	description

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with downloaded data

Exceptions

EXCEPTION	DESCRIPTION
Error 0027	An exception occurs when two objects have to be the same size, but they are not.
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	an exception occurs in when it is not possible to download a file.

EXCEPTION	DESCRIPTION
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.
Error 0009	An exception occurs if the Azure storage account name or the container name is specified incorrectly.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0015	An exception occurs if the database connection has failed.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Import Data](#)

[Export Data](#)

[Import from Web URL via HTTP](#)

[Import from Hive Query](#)

[Import from Azure Table](#)

[Import from Azure Blob Storage](#)

[Import from Data Feed Providers](#)

[Import from On-Premises SQL Server Database](#)

Import from Azure Table

3/10/2021 • 7 minutes to read • [Edit Online](#)

This article describes how to use the [Import Data](#) module in Azure Machine Learning Studio (classic), to import structured or semi-structured data from Azure tables into a machine learning experiment.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The Azure **table** service is a data management service in Azure that can store large amounts of structured, non-relational data. It is a NoSQL data store that accepts authenticated calls from inside and outside Azure.

Importing from Azure table storage requires that you choose one of two account types: a storage account that can be accessed by using a SAS URL, or a private storage account that requires login credentials.

How to import data from Azure tables

Use the Data Import Wizard

The module features a new wizard to help you choose a storage option, select from among existing subscriptions and accounts, and quickly configure all options.

1. Add the [Import Data](#) module to your experiment. You can find the module under **Data Input and Output**.
2. Click **Launch Import Data Wizard** and follow the prompts.
3. When configuration is complete, to actually copy the data into your experiment, right-click the module, and select **Run Selected**.

If you need to edit an existing data connection, the wizard loads all previous configuration details so that you don't have to start again from scratch

Manually set properties in the Import Data module

The following steps describe how to manually configure the import source.

1. Add the [Import Data](#) module to your experiment. You can find this module in the **Data Input and Output** group in the **experiment items** list in Azure Machine Learning Studio (classic).
2. For **Data source**, select **Azure Table**.
3. For **Authentication type**, choose **Public (SAS URL)** if you know that the information has been provided as a public data source. A *SAS URL* is a time bound access URL that you can generate by using an Azure storage utility.
Otherwise choose **Account**.
4. If your data is in a **public** blob that can be accessed by using a SAS URL, you do not need additional credentials because the URL string contains all the information that is needed for download and authentication.

In the **Table SAS URI** field, type or paste the full URI that defines the account and the public blob.

NOTE

In a page accessible via SAS URL, data can be stored using only these formats: CSV, TSV, and ARFF.

5. If your data is in a **private** account, you must supply credentials including the account name and the key.

- For **Table account name**, type or paste the name of the account that contains the blob you want to access.

For example, if the full URL of the storage account is `https://myshared.table.core.windows.net`, you would type `myshared`.

- For **Table account key**, paste the access key that is associated with the storage account.\

If you don't know the access key, see the section, "View, copy and regenerate storage access keys" in this article: [About Azure Storage Accounts](#).

- For **Table name**, type the name of the specific table that you want to read.

6. Choose an option that indicates how many rows the **Import Data** should scan. **Import Data** uses the scan to get the list of columns in the data, and to determine what the column data types should be.

- **TopN:** Scan just the specified number of rows, starting from the top of the dataset.

By default, 10 rows are scanned, but you can increase or decrease that value by using the **Rows count for TopN** option.

If the data is homogenous and predictable, select **TopN** and enter a number for N. For large tables, this can result in quicker reading times.

- **ScanAll:** Scan all rows in the table.

If the data is structured with sets of properties that vary based on the depth and position of the table, choose the **ScanAll** option to scan all rows. This ensures the integrity of your resulting property and metadata conversion.

7. Indicate whether you want the data to be refreshed each time the experiment is run. If you select the **Use cached results** option (the default) the **Import Data** module will read data from the specified source the first time the experiment is run, and thereafter cache the results. If there are any changes to the parameters of the **Import Data** module, the data is re-loaded.

If you deselect this option, the data will be read from the source each time the experiment is run, regardless of whether the data is the same or not.

Examples

For examples of how to use the **Export Data** module, see the [Azure AI Gallery](#).

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Common questions

How can I avoid re-loading the same data unnecessarily?

If your source data changes, you can refresh the dataset and add new data by re-running **Import Data**. However, if you don't want to re-read from the source each time you run the experiment, select the **Use cached results**

option to TRUE. When this option is set to TRUE, the module checks whether the experiment has run previously using the same source and same input options, and if a previous run is found, the data in the cache is used, instead of re-loading the data from the source.

Can I filter data as it is being read from the source?

The [Import Data](#) module does not support filtering as data is being read. The exception is reading from data feeds, which sometimes allow you to specify a filter condition as part of the feed URL.

However, you can change or filter data after reading it into Azure Machine Learning Studio (classic):

- Use a custom R script to change or filter data.
- Use the [Split Data](#) module with a relative expression or a regular expression to isolate the data you want, and then save it as a dataset.

NOTE

If you find that you have loaded more data than you need, you can overwrite the cached dataset by reading a new dataset, and saving it with the same name as the older, larger data.

How does [Import Data](#) handle data loaded from different geographical regions?

If the blob or table storage account is in a different region from the compute node used for the machine learning experiment, data access might be slower. Further, you are charged for data ingress and egress on the subscription.

Why are some characters in my table not displayed correctly?

Azure Machine Learning supports UTF-8 encoding. If your table uses another encoding, the characters might not be imported correctly.

Are there any prohibited characters or characters that are changed during import?

If attribute data contains quotation marks or escaped character sequences, they are handled by using the rules for such characters in Microsoft Excel. All other characters are handled by using the following specifications as a guideline: [RFC 4180](#).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DEFAULT
Data source	List	Data Source Or Sink	Azure Blob Storage	Data source can be HTTP, FTP, anonymous HTTPS or FTPS, a file in Azure BLOB storage, an Azure table, an Azure SQL Database, an on-premises SQL Server database, a Hive table, or an OData endpoint.
Authentication type	PublicOrSas Account	tableAuthType	Account	Specify whether the data is in a public container accessible via SAS URL, or is in a private storage account that requires authentication for access.

Public or SAS - Public storage options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Table URI	any	String		
Rows to scan for property names via SAS	integer			
Rows count for TopN via SAS				

Account - Private storage options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Table account name				
Table account key	any	SecureString		
Table name	any			
Rows to scan for property names	TopN ScanAll			
Rows count for TopN	any	integer		

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with downloaded data

Exceptions

EXCEPTION	DESCRIPTION
Error 0027	An exception occurs when two objects have to be the same size, but they are not.
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	an exception occurs in when it is not possible to download a file.
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.

EXCEPTION	DESCRIPTION
Error 0009	An exception occurs if the Azure storage account name or the container name is specified incorrectly.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Import Data](#)

[Export Data](#)

[Import from Web URL via HTTP](#)

[Import from Hive Query](#)

[Import from Azure SQL Database](#)

[Import from Azure Blob Storage](#)

[Import from Data Feed Providers](#)

[Import from On-Premises SQL Server Database](#)

Import from Azure Blob Storage

3/10/2021 • 17 minutes to read • [Edit Online](#)

This topic describes how to use the [Import Data](#) module in Azure Machine Learning Studio (classic), to read data from Azure Blob Storage, so that you can use the data in a machine learning experiment.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The Azure Blob Service is for storing large amounts of data, including binary data. Azure blobs can be accessed from anywhere, by using either HTTP or HTTPS. Authentication might be required depending on the type of blob storage.

- Public blobs can be accessed by anyone, or by users who have a SAS URL.
- Private blobs require a login and credentials.

Importing from blob storage requires that data be stored in blobs that use the **block blob** format. The files stored in the blob must use either comma-separated (CSV) or tab-separated (TSV) formats. When you read the file, the records and any applicable attribute headings are loaded as rows into memory as a dataset.

NOTE

Import data module does not support connecting to Azure Blob Storage account if "Secure Transfer Required" option is enabled.

For other restrictions on the types of blob storage supported for use with Azure Machine Learning, see the [Technical notes](#) section.

TIP

Need to import data in a format that's not supported? You can use Python or R. See this sample in the Azure AI Gallery: [Load non-text file from Azure Blob Storage](#)

How to import data from Azure blobs

We strongly recommend that you profile your data before importing, to make sure that the schema is as expected. The import process scans some number of head rows to determine the schema, but later rows might contain extra columns, or data that cause errors.

Use the Data Import Wizard

The module features a new wizard to help you choose a storage option, select from among existing subscriptions and accounts, and quickly configure all options.

1. Add the **Import Data** module to your experiment. You can find the module in Studio (classic), in the **Data Input and Output** category.

2. Click **Launch Import Data Wizard** and follow the prompts.
3. When configuration is complete, to actually copy the data into your experiment, right-click the module, and select **Run Selected**.

If you need to edit an existing data connection, the wizard loads all previous configuration details so that you don't have to start again from scratch.

Manually set properties in the Import Data module

The following steps describe how to manually configure the import source.

1. Add the **Import Data** module to your experiment. You can find this module in Studio (classic), in the [Data Input and Output](#) category.
2. For **Data source**, select **Azure Blob Storage**.
3. For **Authentication type**, choose **Public (SAS URL)** if you know that the information has been provided as a public data source. A SAS URL is a time-bound URL for public access that you can generate by using an Azure storage utility.

Otherwise choose **Account**.

4. If your data is in a **public** blob that can be accessed by using a SAS URL, you do not need additional credentials because the URL string contains all the information that is needed for download and authentication.

In the **URI** field, type or paste the full URI that defines the account and the public blob.

NOTE

In a page accessible via SAS URL, data can be stored using only these formats: CSV, TSV, and ARFF.

5. If your data is in a **private** account, you must supply credentials including the account name and the key.

- For **Account name**, type or paste the name of the account that contains the blob you want to access.

For example, if the full URL of the storage account is `https://myshared.blob.core.windows.net`, you would type `myshared`.

- For **Account key**, paste the storage access key that is associated with the account.

If you don't know the access key, see the section, "Manage your Azure storage accounts" in this article: [About Azure Storage Accounts](#).

6. For **Path to container, directory, or blob**, type the name of the specific blob that you want to retrieve.

For example, if you uploaded a file named **data01.csv** to the container **trainingdata** in an account named **mymldata**, the full URL for the file would be:

`https://mymldata.blob.core.windows.net/trainingdata/data01.txt`.

Therefore, in the field **Path to container, directory, or blob**, you would type: `trainingdata/data01.csv`

To import multiple files, you can use the wildcard characters `*` (asterisk) or `?` (question mark).

For example, assuming the container `trainingdata` contains multiple files of a compatible format, you could use the following specification to read all the files starting with `data`, and concatenate them into a single dataset:

trainingdata/data*.csv

You cannot use wildcards in container names. If you need to import files from multiple containers, use a separate instance of the **Import Data** module for each container, and then merge the datasets using the [Add Rows](#) module.

NOTE

If you have selected the option, **Use cached results**, any changes that you make to the files in the container do not trigger a refresh of the data in the experiment.

7. For **Blob file format**, select an option that indicates the format of the data that is stored in the blob, so that Azure Machine Learning can process the data appropriately. The following formats are supported:

- **CSV**: Comma-separated values (CSV) is the default storage format for exporting and importing files in Azure Machine Learning. If the data already contains a header row, be sure to select the option, **File has header row**, or the header will be treated as a data row.

For more information about the CSV format used in Azure Machine Learning, see [Convert to CSV] ([convert-to-csv.md](#))

- **TSV**: Tab-separated values (TSV) is a format used by many machine learning tools. If the data already contains a header row, be sure to select the option, **File has header row**, or the header will be treated as a data row.

For more information about the TSV format used in Azure Machine Learning, see [Convert to TSV](#).

- **ARFF**: This format supports importing files in the format used by the Weka toolset. For more information, see [Convert to ARFF](#).
- **CSV with a specified encoding**: Use this option for CSV files that might have been prepared using a different field separator, or if the source might have used a different character encoding than UTF-8. This format is not supported for files stored in a SAS URL.
- **Excel**: Use this option to read data from Excel workbooks stored in Azure Blob Storage. The Excel format is not supported for files stored in a SAS URL.

8. For CSV files with special encodings, set these additional options to control proper import of the characters:

- **Comma delimiter format**: Choose from a list of common characters used as field separators, including the comma , tab character, and semicolon ;
- **Encoding format**: Choose the character encoding used by the file you want to read. See the [Technical notes](#) section for a list of supported encodings.
- **File has header row**: Select this option if the data already contains a header row. Otherwise, the header is imported as a data row.

9. For Excel files, after specifying the account and container where the Excel file is stored, you must indicate the Excel format and range or table name, using these options:

- **Excel data format**: Indicate whether the data is in an Excel worksheet range, or in an Excel table.
- **Excel sheet or embedded table**: If you select the **Excel sheet** option, specify the name of the worksheet (the tab name) or a table embedded in the workbook. All data from the sheet is read; you cannot specify a range of cells. If you select the **Excel table** option, you must get the table name and not the sheet name, even if there is only one table on a sheet. To view the table name,

click inside the table and then view the **Table Name** property in the **Table Tools** tab.

10. Run the experiment.

Examples

To learn how to use data from Azure Blob Storage in machine learning experiments, see the [Azure Gallery](#):

- [News Categorization sample](#): Loads and then combines multiple datasets from Azure Blob Storage.
- [Student Performance sample](#): Reads data that is stored in the TSV format from Azure Blob Storage.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Common questions

Is there any way of automating data import?

There are a variety of ways to get new data and use it to regularly update an experiment. Much depends on where the source data originates, and the tools you prefer for data movement. see these articles for some ideas.

- [Cheat sheet: automated data pipeline for Azure Machine Learning](#)
- [Use SSIS to move data to Blob storage](#)
- [Retrain experiments programmatically](#)

Automating the execution of the experiment generally requires creation of a web service, which can then be triggered by a task scheduler, PowerShell, or other custom code.

Why did I get an error when I tried to read input data from an existing blob?

There are several possible issues:

- The blob uses an unsupported format
- The account itself was created using an option that is not yet supported by Azure Machine Learning.

Unsupported format: When reading from Azure Blob Storage, currently Azure Machine Learning requires that the blob use the *block blob* format, which lets you upload large blobs efficiently. For example, if you upload a CSV file to blob storage, the file would be stored as a block blob. However, when you create a blob file programmatically, you might be generating a different type of blob, such as the AppendBlob type, which is not supported.

As a workaround, we recommend that you use the **block blob** type.

IMPORTANT

After the blob has been created, the type cannot be changed.

For more information, see [Understanding Block Blobs, Append Blobs, and Page Blobs](#).

Unsupported account type: The import and export modules can read and write data only from Azure storage accounts that were created using the Classic deployment model. In other words, the new Azure Blob Storage account type that offers a hot and cool storage access tiers is not yet supported. Generally, any Azure storage accounts that you might have created before this service option became available should not have been affected.

If you need to create a new account for use with Azure Machine Learning, select **Classic** for the **Deployment model**, or use **Resource manager** and for **Account kind**, select **General purpose** rather than **Blob storage**.

How can I avoid re-loading the same data unnecessarily?

If your source data changes, you can refresh the dataset and add new data by re-running [Import Data](#). However, if you don't want to re-read from the source each time you run the experiment, select the **Use cached results** option to TRUE. When this option is set to TRUE, the module will check whether the experiment has run previously using the same source and same input options, and if a previous run is found, the data in the cache is used, instead of re-loading the data from the source.

Can I filter data as it is being read from the source?

The **Import Data** module does not support filtering as data is being read.

After you have loaded the data into Azure Machine Learning Studio (classic), you can modify the data with these tools:

- Use a custom R script to filter or transform data.
- Use the [Split Data](#) module with a relative expression or a regular expression to isolate the data you want, and then save it as a dataset.

If you find that you have loaded more data than you need, you can overwrite the dataset by reading a new dataset and specify that it be saved with the same name as the older, larger data.

Why does the import process add an extra row at the end of my dataset when it finds a trailing new line?

If the **Import Data** module encounters a row of data that is followed by an empty line or a trailing new line character, an extra row containing missing values is added at the end of the table.

The reason for interpreting a trailing new line as a new row is that **Import Data** cannot determine the difference between an actual empty line and an empty line that is created by the user pressing ENTER at the end of a file.

Because some machine learning algorithms support missing data and would thus treat this line as a case (which in turn could affect the results), you should use [Clean Missing Data](#) to check for missing values, and remove them as needed.

Before you check for empty rows, you might also want to separate the final empty row from other rows with partial missing values, which might represent actual missing values in the source data. To do this, you can divide the dataset by using [Split Data](#). Select the **Select head N rows** option to read all rows but the final row.

What happens if you import data loaded from different geographical regions?

If the blob or table storage account is in a different region from the compute node used for the machine learning experiment, data access might be slower. Further, you are charged for data ingress and egress on the subscription.

Why are some characters in my source file not displayed correctly in the header?

Azure Machine Learning generally supports UTF-8 encoding. If your source file uses another type of encoding, the characters might not be imported correctly.

If you have trouble loading data correctly, try using the option **CSV with encoding** and specify parameters for custom delimiters, the code page, and so forth.

Are there any prohibited characters or characters that are changed during import?

If attribute data contains quotation marks or escaped character sequences, they are handled by using the rules for such characters in Microsoft Excel. All other characters are handled by using the following specifications as a guideline: [RFC 4180](#).

I need to import a very large file. What is the recommended method?

The size limit for uploading local datasets directly to Azure Machine Learning is 1.98 GB. With very large files, adding the dataset to your experiment account can take a long time to complete.

- Estimate 10 minutes or more per GB of data.

- To optimize performance, use a storage account in the same region that Azure ML service uses.

To upload larger files, up to 10 GB, there are several approaches:

- **Use a zipped file.** You can upload datasets to Azure ML Studio (classic) in zipped format, and then use the [Unpack Zipped Datasets](#) module to unpack and save the dataset. Zipped datasets can also be unpacked using the [Execute R Script](#) module but performance might be limited.
- **Use a fast Azure utility such as AzCopy.** Stage the data to Microsoft Azure Blob Storage using a utility such as [AzCopy](#). Then, use the [Import Data](#) module to import data from blob storage to Studio (classic).

For example, the following code shows the AzCopy syntax for writing to blob storage.

```
cd "C:\Program Files (x86)\Microsoft SDKs\Azure\AzCopy"
.\AzCopy.exe /Source:C:\LocalFolder /Dest:https://mystorage.blob.core.windows.net/mycontainer
/DestKey:MyStorageAccountKey /Pattern:myfile.csv
```

I imported a CSV file using a specified encoding but the text is not displayed correctly when I use the option to visualize. Why?

For uploaded datasets, Azure Machine Learning generally supports only UTF-8. However, the [Import Data](#) module supports additional encoding formats. Therefore, after you have imported a file using one of these formats, you might find that the characters are not displaying correctly. The solution is to convert the encoding to UTF-8 using one of these methods:

- Save the imported data as a dataset. (Using a saved dataset instead of the CSV data also might improve performance.)
- If you are using the dataset in the [Execute R Script](#) module, you can enforce the correct coding using script such as this:

```
dataset <- maml.mapInputPort(1)
Encoding(dataset$city) <- "UTF-8"
maml.mapOutputPort("dataset")
```

You can then use **Visualize** on the output of the Execute R Script module and verify that characters are displayed correctly.

What options do I have for importing text files? The CSV is not appropriate for my data.

Processing and cleaning unstructured text to fit neatly in columns is always a challenge. However, if you need to import columns of text data, the TSV format often presents fewer difficulties, though you still need to check for excess tab characters in advance.

We recommend that you review the [Text Classification template](#), in the [Azure AI Gallery](#), to see an example of text ingestion and processing in Azure Machine Learning Studio (classic).

Custom encoding for CSV files

Early versions of the [Import Data](#) module did not support some kinds of valid CSV files. For example, data exported from Excel sometimes contained characters that prevented the file from parsing correctly.

To support a wider possible range of delimiters and character formats, [Import Data](#) now supports choosing the delimiter and encoding format. If you use the **CSV with encoding** option, the result is a more robust and efficient parsing of the CSV file.

You can use the following character encodings:

TYPE	ENCODING
Unicode	Unicode (UTF-8) Unicode Unicode (UTF-32) Unicode (UTF-7)
CJK	Chinese Traditional (Big5) Chinese Simplified (GB2312) Chinese Simplified (Mac) Chinese Simplified (GB2312-80) Chinese Simplified (ISO-2022) Chinese Simplified (GB18030) Japanese (JIS) Korean (ISO) Korean (Mac)
Other	Western European (Windows) Western European (ISO) Hebrew (ISO-Visual) US ASCII

TIP

After the CSV import is complete, we recommend that you save imported files as a dataset to ensure that the imported data uses the UTF-8 encoding in your experiment.

Data type inference in CSV and TSV formats

When the **Import Data** module loads data from a CSV or TSV file in Azure Blob Storage, a type guesser looks for categorical or numerical data in the source file, and represents the discovered type in the metadata for the new dataset.

However, you can override the results of the type guesser by editing the column attributes in the [Edit Metadata](#) module after the data has been loaded.

Module parameters

General options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Data source	List	Data Source Or Sink	Azure Blob Storage	Data source can be HTTP, FTP, anonymous HTTPS or FTPS, a file in Azure BLOB storage, an Azure table, an Azure SQL Database, an on-premises SQL Server database, a Hive table, or an OData endpoint.
Authentication type	PublicOrSas/Account	String	Account	Specify whether the data is in a public container accessible via SAS URL, or is in a private storage account that requires authentication for access.
Use cached results	TRUE/FALSE	Boolean	FALSE	Select to avoid loading data between runs

Public or SAS - Public storage options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
URI	any	String	none	HDFS rest endpoint
File format	ARFF, CSV, or TSV	String	CSV	Select one of the supported formats
URI has header row	Boolean	TRUE/FALSE	TRUE	True if the file contains a header row; if False, the first row of data is used as the column headings

Account - Private storage options

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Account name	any	String	none	Type the name of the storage account
Account key	any	SecureString	none	Paste the account key
Path to container, directory, or blob	any	String	N/A	Type the container or directory name
Blob file format	ARFF, CSV, or TSV	String	CSV	Select one of the supported formats

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
File has header row	any	String	True	Azure storage account name

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with imported data

Exceptions

EXCEPTION	DESCRIPTION
Error 0027	An exception occurs when two objects have to be the same size, but they are not.
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	An exception occurs in when it is not possible to download a file.
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.
Error 0009	An exception occurs if the Azure storage account name or the container name is specified incorrectly.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Import Data](#)

[Export Data](#)

[Import from Web URL via HTTP](#)

[Import from Hive Query](#)

[Import from Azure SQL Database](#)

[Import from Azure Table](#)

[Import from Data Feed Providers](#)

Import from On-Premises SQL Server Database

Import from Data Feed Providers

3/10/2021 • 6 minutes to read • [Edit Online](#)

This article describes how to use the [Import Data](#) module in Azure Machine Learning Studio (classic), to import data provided in the OData format into a machine learning experiment.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Creating an OData endpoint for a data set is one way to make a data model available for consumption via URL. You can also specify which OData operations the endpoint will support. For more information about creating Odata endpoints, see [OData v4 \(ASP.NET\)](#).

How to import data from a feed

We strongly recommend that you profile your data before importing, to make sure that the schema is as expected. The import process scans some number of head rows to determine the schema, but later rows might contain extra columns, or data that cause errors.

Use the Data Import Wizard

The module features a new wizard to help you choose a storage option. Use the wizard to select from among existing subscriptions and accounts, and quickly configure all options.

1. Add the [Import Data](#) module to your experiment. You can find the module in Studio (classic), in the [Data Input and Output](#) category.
2. Click **Launch Import Data Wizard** and follow the prompts.
3. When configuration is complete, to actually copy the data into your experiment, right-click the module, and select **Run Selected**.

If you need to edit an existing data connection, the wizard loads all previous configuration details so that you don't have to start again from scratch.

Manually set properties in the Import Data module

You can also manually configure the import source.

1. Add the [Import Data](#) module to your experiment. You can find this module in Studio (classic), in the [Data Input and Output](#) category.
2. For **Data source**, select **Data Feed Provider**.
3. For **Data content type**, select the type of feed. Currently only OData endpoints are supported.
4. For **Source URL**, paste the URL of a site that provides data in the required format.

For example, the following statement gets the list of products from the Northwind sample database:

```
https://services.odata.org/northwind/northwind.svc/Products
```

For more information, see [OData syntax](#).

5. Select the **Use cached results** option if you don't need to re-load the data after the first time. This is a good option if the data is not expected to change between runs of the experiment.

If there are no other changes to module parameters, the experiment loads the data the first time the module is run, and thereafter uses a cached version of the dataset.

If you need to regularly refresh the data, deselect this option.

6. Run the experiment.

Results

When complete, click the output dataset and select **Visualize** to see if the data was imported successfully.

When [Import Data](#) loads the feed data into Studio (classic), it infers the data type of each column based on the values it contains, either numerical or categorical.

- If a header is present, the header is used to name the columns of the output dataset.
- If there are no existing column headers in the data, new column names are generated using the format
`col1, col2, ..., coln`.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

OData syntax

The query must return a flat table. Flattening nested OData records is not supported.

Some columns included in OData feeds might have data types that are not supported in Studio (classic), such as decimals. You can ingest the data as strings and convert them later using the **Execute R Script** or **Metadata Editor** modules.

For more information about OData syntax and URLs, see [Odata.org - uri conventions](#)

Common questions

Can I filter data as it is being read from the source?

The [Import Data](#) module generally does not support filtering as data is being read. However, you can specify a filter condition as part of the feed resource URL.

To filter data from the feed, use statements supported by the [OData protocol](#). For example, this URL uses the `$filter` expression to get only the orders related to the employee with ID equal to 1.

```
https://services.odata.org/Northwind/Northwind.svc/0rders?$filter=Employee/EmployeeID eq 1
```

For more examples of filter syntax, see [Using Filter Expressions in OData URLs](#).

Alternatively, you can get all the data, and filter it after loading it into Azure Machine Learning Studio (classic):

- Use a custom R script to get only the data you want.
- Use the [Split Data](#) module with a relative expression or a regular expression to isolate the data you want, and then save it as a dataset.

NOTE

If you find that you have loaded more data than you need, you can overwrite the cached dataset by reading a new dataset, and saving it with the same name as the older, larger data.

I get the error, Credentials are required to connect to the OData source. Please refresh and provide credentials to continue. How can I provide credentials?**

The [Import Data](#) module supports only OData endpoints with anonymous access. If the OData service requires credentials, you cannot use the OData option to get the data.

However, if the service is on the same domain, authentication can sometimes happen automatically without any user input.

As a workaround, you can use PowerQuery or PowerPivot to read feed data and then get the data from Excel.

How can I avoid re-loading the same data unnecessarily?

If your source data changes, you can refresh the dataset and add new data by re-running [Import Data](#). However, if you don't want to re-read from the source each time you run the experiment, select the **Use cached results** option to TRUE. When this option is set to TRUE, the module will check whether the experiment has run previously using the same source and same input options, and if a previous run is found, the data in the cache is used, instead of re-loading the data from the source.

Why do I get an error message "Type Decimal is not supported"?

The `decimal` data type is not supported in Azure Machine Learning. The reason is that [Import Data](#) cannot automatically perform any conversions that would result in a loss of precision.

For more information about supported data types, see [Module Data Types](#).

As a workaround, you can read the data as a string data type, and then use [Edit Metadata](#) to convert the decimals to a supported data before reading the data.

Why are some characters in the feed not displayed correctly?

Azure Machine Learning supports the UTF-8 encoding. If your source uses another type of encoding, the characters might not be imported correctly.

As a workaround, you can save the data to a CSV file in Azure table storage or Azure blob storage. Then, use the option **CSV with encoding** to specify parameters for custom delimiters, the code page, and so forth.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Data source	List	Data Source Or Sink	Azure Blob Storage	Data source can be HTTP, FTP, anonymous HTTPS or FTPS, a file in Azure BLOB storage, an Azure table, an Azure SQL Database, an on-premises SQL Server database, a Hive table, or an OData endpoint.
Data content type	List (subset)	Url Contents	OData	Data format type
Source URL	any	String		URL for Power Query data source
Use cached results	TRUE/FALSE	Boolean	FALSE	description

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with downloaded data

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	an exception occurs in when it is not possible to download a file.
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Import Data](#)

[Export Data](#)

[Import from Web URL via HTTP](#)

[Import from Hive Query](#)

[Import from Azure SQL Database](#)

[Import from Azure Table](#)

[Import from Azure Blob Storage](#)

[Import from On-Premises SQL Server Database](#)

Import from On-Premises SQL Server Database

3/10/2021 • 8 minutes to read • [Edit Online](#)

This article describes how to use the [Import Data](#) module in Azure Machine Learning Studio (classic), to import data from an on-premises SQL Server database into a machine learning experiment.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Azure Machine Learning can access an on-premises SQL Server database if the data is provided using a Microsoft Data Management Gateway. Therefore, before you use [Import Data](#), you must meet these requirements:

- Install a [Microsoft Data Management Gateway](#) that can access the data source
- Register the gateway in your Azure Machine Learning workspace
- Configure the [Import Data](#) to identify the gateway

After the gateway connection is established, you can then specify additional properties, such as the server and database names, authentication method, and a database query.

How to install a Microsoft Data Management Gateway

To access an on-premises SQL Server database in Azure Machine Learning, you need to download and install the [Microsoft Data Management Gateway](#), and then register the gateway in Machine Learning Studio (classic).

For details about installing and registering the gateway, see these articles:

- [Perform advanced analytics with Azure Machine Learning using data from an on-premises SQL Server database](#)
- [Troubleshoot issues with using Data Management Gateway](#)

How to import data from an on-premises SQL Server database

After a Data Management Gateway has been installed on a computer where it can access your SQL Server database, and you have registered the gateway in Machine Learning Studio (classic), you must configure the [Import Data](#) module.

Before you start, disable your browser's pop-up blocker for the site, studio.azureml.net.

If you are using the Google Chrome browser, you must download and install one of the plug-ins that are available at the Google Chrome WebStore: [Click Once App Extension](#).

Use the Data Import Wizard

The module features a new wizard to help you choose a storage option, select from among existing subscriptions and accounts, and quickly configure all options.

1. Add the [Import Data](#) module to your experiment. You can find the module in Studio (classic), in the

Data Input and Output category.

2. Click **Launch Import Data Wizard** and follow the prompts.
3. When configuration is complete, to actually copy the data into your experiment, right-click the module, and select **Run Selected**.

If you need to edit an existing data connection, the wizard loads all previous configuration details so that you don't have to start again from scratch.

Manually set properties in the Import Data module

1. Add the [Import Data](#) module to your experiment. You can find the module in Studio (classic), in the **Data Input and Output** category.
2. For **Data source**, select **On-Premises SQL Database**.
3. Set the following options specific to the SQL Server database.
 - **Data gateway**: Select the gateway you created. The gateway must be registered or it does not show in the list.
 - **Database server name**: Type the name of the SQL Server instance.
 - **Database name**: Type the database name.
 - Click **Enter values** under **User name and password** and enter your database credentials. You can use Windows Integrated Authentication or SQL Server Authentication depending upon how your on-premises SQL Server is configured.

IMPORTANT

The credential manager must be launched from within the same network as the SQL Server instance and the gateway client. Credentials cannot be passed across domains.

- Type or paste into **Database query** a SQL statement that describes the data you want to read. Always validate the SQL statement and verify the query results beforehand, using a tool such as Visual Studio Server Explorer or SQL Server Data Tools.
 - If the dataset is not expected to change between runs of the experiment, select the **Use cached results** option. When this is selected, if there are no other changes to module parameters, the experiment will load the data the first time the module is run, and thereafter use a cached version of the dataset.
4. Run the experiment.

Results

As [Import Data](#) loads the data into Studio (classic), some implicit type conversion might be performed, depending on the data types used in the source database. For more information about data types, see [Module Data Types](#).

When complete, click the output dataset and select **Visualize** to see if the data was imported successfully.

Optionally, you can change the dataset and its metadata using the tools in Studio (classic):

- Use [Edit Metadata](#) to change column names, convert a column to a different data type, or to indicate which columns are labels or features.
- Use [Select Columns in Dataset](#) to select a subset of columns.

- Use [Partition and Sample](#) to separate the dataset by criteria, or get the top n rows.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Common questions

Can I filter data as it is being read from the source?

The [Import Data](#) module itself does not support filtering as data is being read. We recommend that you create a view or define a query that generates only the rows you need.

NOTE

If you find that you have loaded more data than you need, you can overwrite the cached dataset by reading a new dataset, and saving it with the same name as the older, larger data.

Why do I get the error, "Type Decimal is not supported"

When reading data from a SQL database, you might encounter an error message reporting an unsupported data type.

If the data you get from the SQL database includes data types that are not supported in Azure Machine Learning, you should cast or convert the decimals to a supported data type before reading the data. The reason is that [Import Data](#) cannot automatically perform any conversions that would result in a loss of precision.

Why are some characters not displayed correctly

Azure Machine Learning supports the UTF-8 encoding. If string columns in your database use a different encoding, the characters might not be imported correctly.

One option for preserving these characters is to export the data to a CSV file in Azure storage, and use the option **CSV with encoding** to specify parameters for custom delimiters, the code page, and so forth.

I set up a Data Management Gateway on my on-premises server. Can I share the same gateway between workspaces

No. You must create a separate gateway for each workspace.

While you can set up multiple Data Management Gateways in a single workspace (for example, one each for development, testing, production, etc.), a gateway can't be shared across workspaces.

I have set up a Data Management Gateway on my on-premises server that I use for Power BI or Azure Data Factory and want to use the same gateway for Azure Machine Learning

Each service requires a separate Data Management Gateway. If you already have a gateway that's being used for Power BI or Azure Data Factory, you must set up a separate server and install a gateway for machine learning.

You can't install multiple gateways on a single server.

I want to be able to export data to my on-premises SQL server. Can I use the gateway with the Export Data module to write data to my on-premises SQL server?

Currently, Azure Machine Learning only supports importing data. We're evaluating whether you'll be able to write to your on-premises database in the future. In the meantime, you can use Azure Data Factory to copy data from the cloud to your on-premises database.

I have a data source that is not Microsoft SQL Server (Oracle, Teradata, etc.). Can I read the data in Azure Machine Learning using the on-premises option in the Import Data module?

Currently the Azure Machine Learning [Import Data](#) module supports only Microsoft SQL Server.

As a workaround, you can use Azure Data Factory to copy your on-premises data into cloud storage such as Azure Blob Storage or Azure Database, and then use your cloud data source in the [Import Data](#) module.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Data source	List	Data Source Or Sink	Azure Blob Storage	Data source can be HTTP, FTP, anonymous HTTPS or FTPS, a file in Azure BLOB storage, an Azure table, an Azure SQL Database, an on-premises SQL Server database, a Hive table, or an OData endpoint.
Data gateway	any	DataGatewayName	none	Data gateway name
Database server name	any	String	none	On-premises SQL Server
Database name	any	String	none	On-premises SQL Server database instance
User name and password	any	SecureString	none	User name and password
Database query	any	StreamReader	none	On-premises SQL query

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with downloaded data

Exceptions

EXCEPTION	DESCRIPTION
Error 0027	An exception occurs when two objects have to be the same size, but they are not.
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0030	An exception occurs in when it is not possible to download a file.
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.

EXCEPTION	DESCRIPTION
Error 0048	An exception occurs when it is not possible to open a file.
Error 0015	An exception occurs if the database connection has failed.
Error 0046	An exception occurs when it is not possible to create a directory on specified path.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

- [Import Data](#)
- [Export Data](#)
- [Import from Web URL via HTTP](#)
- [Import from Hive Query](#)
- [Import from Azure SQL Database](#)
- [Import from Azure Table](#)
- [Import from Azure Blob Storage](#)
- [Import from Data Feed Providers](#)

Import from Azure Cosmos DB

3/10/2021 • 8 minutes to read • [Edit Online](#)

This article describes how to use the [Import Data](#) module in Azure Machine Learning Studio (classic), to import data from Azure Cosmos DB for use in a machine learning experiment.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Azure Cosmos DB supports NoSQL database storage, using a flexible data model. The advantages of using the SQL APIs in this data store for machine learning include fast and predictable performance, automatic scaling, global distribution, and rich query capabilities.

Together with Azure SQL Database, this option lets you dynamically filter incoming datasets.

Learn how it works: [Learn about Azure Cosmos DB](#)

- To get started with machine learning using data from Azure Cosmos DB, you must have access to an existing Azure Cosmos DB account containing a collection of related documents.

NOTE

The user interface in Azure Machine Learning Studio (classic) still uses the name DocumentDB in many places. Therefore, you may continue to see references to DocumentDB, even though the API has been incorporated into Azure Cosmos DB.

How to use Import Data with Azure Cosmos DB

We strongly recommend that you profile your data before importing, to make sure that the schema is as expected. The import process scans some number of head rows to determine the schema, but later rows might contain extra columns, or data that cause errors.

Import data using the wizard

The module features a new wizard to help you choose a storage option, select from among existing subscriptions and accounts, and quickly configure all options.

1. Add the **Import Data** module to your experiment. You can find the module under **Data Input and Output**.
2. Click **Launch Import Data Wizard** and follow the prompts.
3. When configuration is complete, to actually copy the data into your experiment, right-click the module, and select **Run Selected**.

TIP

If you need to edit an existing data connection, the wizard loads all previous configuration details. You don't have to start again from scratch.

Manually set properties in the Import Data module

The following steps describe how to manually configure the import source.

1. Add the [Import Data](#) module to your experiment. You can find this module in the [Data Input and Output](#) category.
2. For **Data source**, select **Azure DocumentDB**.

You might need to provide connection information for the document database.

TIP

Look for the name of the option in Machine Learning Studio (classic) to change at a later date. The import functionality was not affected by the name change.

3. For **Endpoint URL**, in the Azure Portal, click **Keys**, and copy the contents of the **URI** field at the top of the page.
4. For **Database ID**, paste the name of the database to use.

To get the database name from the Azure Portal, click **Document Explorer**. You can view the list of databases and collections in this pane.

5. For **DocumentDB Key**, paste in an access key for the account.

To locate the keys, click **Keys**, and then copy the content of either the **PRIMARY KEY** or **SECONDARY KEY** fields.

6. For **Collection ID**, type the name of the collection as shown in the specified CosmosDB database.

7. Define a SQL query and filter condition on the data, by using the [SQL query](#) and [SQL query parameters](#) options.

For **SQL query**, type a query that defines the data to retrieve from the collection. We recommend that you use the [Query Explorer](#) to create and test your CosmosDB queries beforehand.

For **SQL query parameters**, provide an expression in JSON format that can be used to dynamically filter the data returned. Typically you supply the actual value of the parameter value when running the experiment as part of a Web service.

If you use a parameter, you must define the filter variable name as part of the WHERE clause specified in the **SQL query** text box.

If you do not specify a filter expression, by default, the value is set to "{}", and all records are returned.

See the [Technical Notes](#) section for examples, known issues, and additional advice about SQL queries on CosmosDB.

8. Select the **Use cached results** option if you want to reuse existing results.

If you deselect this option, the data is read from the source each time the experiment is run, regardless of whether the data is the same or not.

Azure Machine Learning **cannot** compare the cached data against the data in your CosmosDB account. Hence, there is no way to perform incremental updates from Azure Machine Learning.

If you want to re-import only when the data changes, you must define that logic in another application, such as Azure Data Factory. For more information, see [Move data to and from Azure Cosmos DB using Azure Data Factory](#).

9. Run the experiment, or select just the **Import Data** module and click **Run selected**.

Results

After you have run the module or experiment, you can right-click the output of the module to visualize the results in tabular format.

To capture a snapshot of this data in your Azure Machine Learning workspace as a dataset, you can right-click the module's output and select **Save As Dataset**. However, doing so captures only the data available at the time of import. If the data is expected to change frequently, rerun **Import Data** as needed.

Examples

For a detailed walkthrough of how to use Azure Cosmos DB as a data source for machine learning, see the [Azure AI Gallery](#).

- [SQL queries for Azure Cosmos DB](#): This article explains how to perform SQL queries on Azure Cosmos DB data.

Technical notes

This section contains advanced configuration options and answers to commonly asked questions.

Examples of simple and parameterized queries

Suppose you want to use only the data on volcanoes with elevations under 10000 feet.

Simple query

Paste the following query in the **SQL query** text box:

```
Select * from volcanodb where volcanodb.Elevation < 10000
```

In this case, the value of the filter expression is set to "{}", and all records are returned.

Parameterized query

To get only the volcano data related to a specific country, you can specify the country value as a parameter passed to the query at run time. This requires these changes:

1. In the **SQL query** text box, define a variable to apply to the **Country** field as part of the SQL query:

```
Select * from volcanodb where volcanodb.Country = @param1
```

2. In the **SQL query parameters** text box, specify the parameter name and its value in JSON format, like this:

```
{"@param1": "Turkey"}
```

Resources

If you don't have an existing document store, see these articles to get started.

- [Azure Cosmos DB: Build a SQL API web app with .NET and the Azure portal](#)
- [Quickstart: Build a .NET console app to manage Azure Cosmos DB SQL API resources](#)
- [Azure Cosmos DB Migration Tool](#)

Data migration and query syntax help

For samples of queries on a JSON data store, download the [Azure Cosmos DB query cheat sheet](#).

If you need to upload content into Azure Cosmos DB, we recommend the [Azure Cosmos DB migration tool](#). It validates, uploads, and indexes your data. The tool supports multiple sources, including MongoDB, Amazon DynamoDB, HBase, SQL Server databases, and CSV files.

Using schema-less queries

If the data is consistent and predictable, you can use straightforward SQL-like syntax, such as

`SELECT * FROM <document collection>`. This is called a *schema-less query* because you have not named the exact attributes to return. Such a query would return all the fields and all the rows from the specified collection.

However, not specifying a schema can lead to unexpected results or a run-time error if the documents have inconsistent schemas. This is because the [Import Data](#) module attempts to infer the schema based on a predetermined number of rows as follows:

1. When no attributes are specified, the module scans the first row in the CosmosDB database.
2. The module creates column names based on attributes, and guesses what the column data types should be based on the example row.
3. If later rows contain any new or different attributes, a run-time error is generated.

Therefore, we recommend that you always specify the attributes and values to return from the CosmosDB data store. For example, rather than use the `SELECT *` syntax, we recommend that you name all attributes retrieved by the query, like this:

```
SELECT MyTable.Gender, MyTable.Age, MyTable.Name FROM <document collection>
```

Module parameters

The following table includes only those parameters for the [Import Data](#) module that are applicable to the Azure Cosmos DB option.

NAME	RANGE	TYPE	REQUIRED	DEFAULT	DESCRIPTION
Data source	list	HTTP	required	none	Data source can be HTTP, FTP, anonymous HTTPS or FTPS, a file in Azure BLOB storage, an Azure table, an Azure SQL Database, a Hive table, an OData endpoint, or Azure Cosmos dB.
Endpoint URL	any	string	required	none	Provide the URI for the Azure Cosmos DB server
Database ID	any	string	required	none	Provide the name of the Azure Cosmos DB database
DocumentDB Key	any	SecureString	required	none	Provide a valid API key for the Azure Cosmos DB account

NAME	RANGE	TYPE	REQUIRED	DEFAULT	DESCRIPTION
Collection ID	any	string	required	none	Provide the name of a collection in the Azure Cosmos DB database
SQL Query	any	string	required	none	A SQL query specifying records to return from the Azure Cosmos DB data store

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with downloaded data

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0029	An exception occurs when an invalid URI is passed.
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type to the type required by the target method.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Import Data](#)

[Export Data](#)

[Import from Web URL via HTTP](#)

[Import from Hive Query](#)

[Import from Azure SQL Database](#)

[Import from Azure Blob Storage](#)

[Import from Data Feed Providers](#)

[Import from On-Premises SQL Server Database](#)

Load Trained Model

3/10/2021 • 5 minutes to read • [Edit Online](#)

Load a web-hosted trained model

Category: [Data Input and Output](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Load Trained Model** module in Azure Machine Learning Studio (classic), to load an already trained model for use in an experiment.

This module requires an existing trained model. Typically, you create and then train the model in a different experiment, and then save the model either to your workspace, or to one of the supported cloud storage options.

Then, you use the **Load Trained model** module to get the trained model and run it in a new experiment.

How to use Load Trained Model

To use an existing model to make predictions for new data:

- The model must have previously been trained and then saved in the iLearner format.
- The model must be accessible either by URL or in Azure blob storage.

This section describes how to save a model, get a saved model, and apply a saved model.

Save a trained model

You can save models by using the Studio (classic) interface, or using an experiment that runs as a web service.

Save a model using a web service

1. Create an experiment that does the training or retraining of the model as a web service
2. Publish that experiment as a Web service.
3. When you call the BES endpoint of the training web service, the Web service saves a trained model using the iLearner interface and saves the file in the Azure blob storage account that you specify.

For step-by-step information about how to create a training web service, see these articles:

- [Retrain Machine Learning Models Programmatically](#)
- [Saving an experiment as a Web service](#)

Save a model in Studio (classic)

1. Run the experiment that builds and trains the model.
2. When training is complete, right-click the module that was used for training, select **Trained model**, and then click **Save as trained model**.

3. By default, models are saved to your Studio (classic) workspace. You can view them using the Studio (classic) UI.

The following modules can create a saved model that uses the required [iLearner](#) interface:

- [Train Model](#)
- [Train Clustering Model](#)
- [Train Anomaly Detection Model](#)
- [Tune Model Hyperparameters](#)
- [Sweep Clustering](#)

NOTE

Arbitrary models are not supported; the model must have been saved in the default binary format used for persisting Azure Machine Learning models.

Load the model into a new experiment

1. Add the **Load Trained Model** module to your experiment in Studio (classic).
2. For **Data source**, indicate the location of the trained model, using one of the following options:
 - **Web URL via HTTP**: Provide a URL that points to the experiment and the file representing the trained model. In Azure Machine Learning, trained models are by default saved in the [iLearner](#) format.
 - **Azure Blob Storage**: Select this option only if you exported the trained model to Azure storage. You must then provide the account name and account key, and the path to the container, directory, or blob.
3. If you intend to create a Request-Response web-service that is based on the current experiment, select the option, **Allow to use in RRS**. Otherwise, scoring is performed using the Batch Execution Service (BES) option, which is recommended. See the [Technical notes](#) section for details.
4. Select the **Use cached results** option if you want to load the trained model from cache, when the cache is available and populated. This option is ignored after the experiment is deployed as a Web service API.

Examples

For examples of how to use this module, see the [Cortana Intelligence Gallery](#).

- **Load a Trained Deep Learning Model**: The example creates a custom neural network for image detection. By using the **Load Trained Model** module, you can easily re-use this model without having to train it, which can be time-consuming.

This collection includes a training experiment, to create the model, and a predictive experiment, in which the model is loaded as a web service and used for predictions.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Common questions

Why is RRS use not enabled by default

It is generally expected that RRS calls return results within a short period of time. However, because the module must load the trained model in the form of a blob from an Azure storage account or a file hosted on a public HTTP endpoint, file operations might introduce unpredictable delays.

Therefore, we generally advise that the Web service be run in batch execution mode (BES). If you select the option for .execution using RRS, be aware of the potential for delay. For general information about execution times, see the [Azure Machine Learning SLA](#).

Does the trained model load faster if I use the cached results option

Yes, but only when the experiment is run in Azure Machine Learning Studio (classic), and only after the cache has been filled by the first run. After the experiment is deployed as web service, this flag is ignored by web service execution.

Is there a way to automate the process

You can use PowerShell to simplify or automate many tasks in Azure Machine Learning. For example, you can download the contents of an entire experiment or a particular module, export the definition of web service, or invoke the web service execution API. For more information, see [PowerShell Module for Microsoft Azure Machine Learning](#).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Allow to use in RRS	True/False	Boolean	false	Allow this module to run in request-response web service, which may incur unpredictable delays
Data source	Web URL via HTTP, or Azure Blob Storage	T_DataSourceOrSink	Azure Blob Storage	Data source can be HTTP or a file in Azure blob storage (required)
For Web URL via HTTP:				
Data source URL	any	String		URL for HTTP
For Azure Blob Storage:				
Account Name	any	String		Account name
Account key	any	SecureString		Key associated with the Windows Azure Storage account
Path to container or directory or blob	any	String		Path to blob or name of table

Outputs

NAME	TYPE	DESCRIPTION
Trained Model	ILearner interface	Trained model

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Data Input and Output](#)

Unpack Zipped Datasets

3/10/2021 • 6 minutes to read • [Edit Online](#)

Unpacks datasets from a zip package in user storage

Category: [Data Input and Output](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Unpack Zipped Datasets** module in Azure Machine Learning Studio (classic), to upload data and script files in compressed format, and then unzip them for use in an experiment.

The purpose of this module is to reduce data transfer times when working with very large datasets by saving and uploading your data files in a compressed format. Generally, zipping files is a good option when your dataset is so large that you want to use compression for the upload, to minimize upload time and associated costs.

The module takes as input a dataset in your workspace. The dataset must have been uploaded in a compressed format. The module then decompresses the dataset and adds the data to your workspace.

How to use Unpack Zipped Datasets

This section describes how to prepare your data and then unzip it in Azure Machine Learning Studio (classic).

Step 1. Prepare files

Before uploading your file, make sure that the data in the file can be used in Azure Machine Learning:

- Ensure that the data in the file uses UTF-8 encoding.
If the file is small enough, you can open it in Notepad and then save the file in the desired encoding. Many other text editors offer similar functionality. For CSV files, you can use Excel's **Save As** or **Export** commands to specify a file format and encoding.
- Verify that the data files use a supported **format**, such as CSV, TSV, ARFF, or SVMLight.
- Compress the data by adding the data file to a **.ZIP** or **.GZ** format archive file. Other archive types are not supported.
- Remove password protection. If any of the files or the compressed folder itself has been encrypted or password-protected, you must unlock or decrypt the file before you upload it. The module cannot detect encrypted data types and does not support dialog boxes for password entry from arbitrary clients.

Step 2. Upload dataset to your workspace

Next, upload the zipped dataset to your experiment workspace.

1. Click **NEW**, select **DATASET**, and select **FROM LOCAL FILE**.

2. Locate the zipped file to upload. When you select the file, the type should automatically be set to **Zip file (.zip)**.

Step 3. Add zipped dataset to experiment

After the dataset has uploaded completely, add it to your experiment in zipped format.

1. In the left-hand navigation pane of Azure Machine Learning Studio (classic), select **Saved Datasets**, and then expand **My Datasets**.
2. Locate the zipped dataset that you just uploaded, and drag it to the experiment canvas.

Step 4. Unpack dataset

The final step is to unpack the dataset.

1. Connect the zipped dataset to the input of the **Unpack Zipped Datasets** module.
2. In **Dataset to Unpack**, type the name of a single dataset to unpack.
 - If you saved a worksheet with the name **Sheet1** as an Excel CSV file named **Test.csv**, the name of the dataset would be **Test.csv**, not **Sheet1**.
 - The name that you type in the **Dataset to Unpack** text box must be exactly the same as the name of the original file **before** it was compressed, including the file name extension. For example, if you want to unpack a dataset based on the text file **Users.txt**, type **Users.txt**, not **Users**.
 - If you put multiple files into one compressed folder, you must unpack one dataset at a time.

TIP

If you leave the property blank, the module gets the file name from the zipped file, assuming the compressed archive file contains only one source file. If the compressed archive contains multiple files, a run-time error is raised.

3. For **Dataset file format**, specify the original format of the dataset: that is, the format before it was zipped.

You can upload and unzip datasets that were created using any of these formats: CSV, ARFF, TSV, SvmLight.

If this property is left empty, the module identifies the dataset using the source file name.

4. Select the option, **File has header row**, if the original dataset had a header row. Otherwise the first row of data is used as the header. If this is not what you want, add a header prior to input.

This option applies only to .CSV and .TSV files.

NOTE

If you change the format of the file, this option is reset.

5. If the file is compressed, use the **Compression file format** option to specify the algorithm that was used to compress or expand the file.

Currently the .ZIP and GZ (or Gzip) formats are supported.

6. Run the experiment.

Results

- To verify that the data was imported correctly, right-click the **Unpacked Zipped Datasets** module, and select **Visualize**.
- To change the name of the dataset, right-click the **Unpacked Zipped Datasets** module, and select **Save as Dataset**. At this point you can type a different name.

This option is handy if you are unpacking multiple datasets from a single ZIP file.

Examples

To demonstrate how this module works, we created a sample .ZIP file containing four different CSV files. All files were saved from Excel.

FILE NAME	DESCRIPTION
names-uni.csv	Unicode file with column headings
names-utf.csv	UTF-8 file with column headings
nonames-uni.csv	Unicode file with no column headings
nonames-utf8.csv	UTF-8 file with no column headings

The entire zipped file was uploaded, and then the **Unpack Zipped Datasets** module was run four times to extract each of the four files, using these settings:

1. **Dataset to unpack** = names-uni.csv, **File has header row** = TRUE
2. **Dataset to unpack** = names-utf8.csv, **File has header row** = TRUE
3. **Dataset to unpack** = nonames-uni.csv, **File has header row** = FALSE
4. **Dataset to unpack** = nonames-utf8.csv, **File has header row** = FALSE

The results were as expected:

FILE NAME	UPLOAD RESULT
names-uni.csv	Error 0049: Error while parsing the file. File is not Unicode (UTF-8) encoded
names-utf8.csv	Success. Uses original column names from source file.
nonames-uni.csv	Error 0049: Error while parsing the file. File is not Unicode (UTF-8) encoded
nonames-utf8.csv	Success. Column names Col1, col2, ...coln are automatically added to the dataset.

NOTE

If you use the option, **File has header row** = TRUE, and the source file actually does not have a column heading, the first row of data is used as the column heading.

Technical notes

You cannot use this module to unpack zipped R packages into your workspace. R packages must be uploaded

and consumed as zipped files.

For more information about how to work with zipped R packages, see [Execute R Script](#).

NOTE

Confused about the difference between UTF-8 and Unicode? See this Wikipedia article: [What is UTF-8](#)

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Compression file format	Zip Gzip	compression rule	Zip	Compression algorithm used to compress or expand the file.
Dataset to Unpack	Any	String	none	Name of dataset to register with Azure ML Studio (classic). If the name of a dataset is not specified, the name is obtained from the file name in the zipped file.
Dataset file format	CSV TSV ARFF SVMLIGHT	File format	CSV	File format of the dataset in the zipped file
File has header row	TRUE/FALSE	Boolean	False	Set to True only if the CSV/TSV file has a header row

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Zip	Zipped file containing datasets

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Output dataset

See also

[Data Input and Output](#)

Data Transformation

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article lists the modules that are provided in Azure Machine Learning Studio (classic) for data transformation. For machine learning, *data transformation* entails some very general tasks, such as joining datasets or changing column names. But, it also includes many tasks that are specific to machine learning, such as normalization, binning and grouping, and inference of missing values.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

IMPORTANT

Data that you use in Machine Learning Studio (classic) is generally expected to be "tidy" before you import it to Machine Learning Studio (classic). Data preparation might include, for example, ensuring that the data uses the correct encoding and checking that the data has a consistent schema.

You can use Azure Machine Learning Workbench to transform and prepare all kinds of data. For examples, see [Data transformations "by example" in Machine Learning Workbench](#).

Modules for data transformation are grouped into the following task-based categories:

- [Creating filters for digital signal processing](#): Digital signal filters can be applied to numeric data to support machine learning tasks such as image recognition, voice recognition, and waveform analysis.
- [Generating and using count-based features](#): Count-based featurization modules help you develop compact features to use in machine learning.
- [General data manipulation and preparation](#): Merging datasets, cleaning missing values, grouping and summarizing data, changing column names and data types, or indicating which column is a label or a feature.
- [Sampling and splitting datasets](#): Divide your data into training and test sets, split datasets by percentage or by a filter condition, or perform sampling.
- [Scaling and reducing data](#): Prepare numerical data for analysis by applying normalization or by scaling. Bin data into groups, remove or replace outliers, or perform principal component analysis (PCA).

List of modules

The following module categories are included in the **Data Transformation** category:

- [Data Transformation - Filter](#)
- [Learning with Counts](#)
- [Data Transformation - Manipulation](#)
- [Data Transformation - Sample and Split](#)
- [Data Transformation - Scale and Reduce](#)

See also

- Data Format Conversions
- Data Input and Output
- Feature Selection
- Module categories and descriptions

Data Transformation - Filter

3/10/2021 • 5 minutes to read • [Edit Online](#)

This article describes how you can use the filter modules in Azure Machine Learning Studio (classic) to transform digital data. The modules in this group of tools for Machine Learning Studio (classic) are based on filters that were developed for digital signal processing technology.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Filters typically are applied to data in the data processing stage or the preprocessing stage. Filters enhance the clarity of the signal that's used for machine learning. For example, you can use the filter modules in Machine Learning Studio (classic) for these processing tasks:

- Clean up waveforms that are used for speech recognition.
- Detect trends or remove seasonal effects in noisy sales or economic data.
- Analyze patterns or artifacts in telemetry signals.

These modules provide easy configuration of filters by using well-researched algorithms to mathematically transform waveform data. You can also create a custom filter if you have already determined the correct coefficients to apply to your data.

Related tasks

If you need to do tasks such as excluding data from a dataset on a row-by-row basis, removing missing values, or reducing the size of a dataset, use these modules instead:

- [Clean Missing Data](#): Remove missing values, or replace missing values with placeholders.
- [Partition and Sample](#): Divide or filter your dataset by using criteria such as a range of dates, a specific value, or regular expressions.
- [Clip Values](#): Set a range of values, and keep only the values within that range.

Filters in digital signal processing

Just like you can attach a filter to a camera to compensate for lighting or to create special effects, you can apply a filter to the data that you use for machine learning. Filters can help improve the clarity of a signal, capture interesting characteristics, or reduce noise.

The ideal filter would eliminate all noise and have uniform sensitivity for the desired signal. But, designing even a pretty good filter might take many iterations or combinations of techniques. If you succeed in designing an effective filter, consider saving the filter so that you can reuse it when you transform new data.

In general, filtering is based on the principles of *waveform analysis*. When you design a filter, you look for ways to suppress or amplify parts of the signal, to expose underlying trends, to reduce noise and interference, or to identify data values that otherwise might not be perceived.

Various techniques are applied to decompose individual trends or waveform components that create actual data

values. The series of values can be analyzed by using trigonometric functions to identify and isolate individual waveforms. (This is true whether it's an econometric series or the composite frequencies of audio signals.) Filters can then be applied to these waveforms to eliminate noise, amplify some waves, or remove targeted components.

When filtering is applied to a noisy series to isolate different components, you can specify which frequencies to remove or strengthen by specifying the band of frequencies to work with.

Digital filters in Machine Learning Studio (classic)

The following types of filters are supported in Machine Learning Studio (classic):

- **Filters based on waveform decomposition.** Examples include finite impulse response (FIR) and infinite impulse response (IIR) filters. These filters work by removing specific components from an overall series. You can then view and investigate the simplified waveform.
- **Filters based on moving averages or median values.** These filters smooth out variations in a data series by averaging across windows of time. The windows can be fixed or sliding, and can have different shapes. For example, a triangular window peaks at the current data point (weights the current value stronger) and tails off before and after the data point (weights preceding and following values less strongly).
- **User-defined or custom filters.** If you already know the transformations that should be applied to a data series, you can create a user-defined filter. You provide the numeric coefficients that are applied to transform the data series. A custom filter can emulate an FIR or IIR filter. However, with a custom filter, you have more control over the values to apply at each point in the series.

Filter terminology

The following list includes simple definitions of terms that are used in the parameters and properties of filters:

- **Passband:** The range of frequencies that can pass through a filter without being attenuated or weakened.
- **Stopband:** A range of frequencies between specified limits through which signals are not passed. You define the stopband by setting cut-off frequencies.
- **High pass:** Let only high frequencies through.
- **Low pass:** Accept only frequencies below a specified cut-off value.
- **Corner:** Defines the boundary between the stopband and passband frequencies. Typically, you have the option to decide whether the corner is included in or excluded from the band. A first-order filter causes gradual attenuation until the corner frequency. After that, the filter causes exponential attenuation. Higher-order filters (such as Butterworth and Chebyshev filters) have steeper slopes after the corner frequency. Higher-order filters attenuate the values in the stopband much more rapidly and fully.
- **Bandstop filter** (also called a *band reject* filter or a *notch* filter): Has only one stopband. You define the stopband by specifying two frequencies: the high cut-off frequency and the low cut-off frequency. A *bandpass* filter typically has two stopbands: one on either side of the desired component.
- **Ripple:** A small, unwanted variation that occurs periodically. In Machine Learning, you can specify the amount of ripple to tolerate as part of the parameters in the IIR filter design.

TIP

Need more information? If you are new to digital signal processing, see [An Introduction to Digital Signal Processing](#). The website provides definitions and helpful visual aids that explain basic terminology and concepts.

List of modules

The following modules are included in the **Data Transformation - Filter** category:

- [Apply Filter](#): Applies a filter to specified columns of a dataset.
- [FIR Filter](#): Creates an FIR filter for signal processing.

- [IIR Filter](#): Creates an IIR filter for signal processing.
- [Median Filter](#): Creates a median filter that's used to smooth data for trend analysis.
- [Moving Average Filter](#): Creates a moving average filter that smooths data for trend analysis.
- [Threshold Filter](#): Creates a threshold filter that constrains values.
- [User-Defined Filter](#): Creates a custom FIR or IIR filter.

See also

- [Data Transformation](#)
- [A-Z module list](#)

Apply Filter

3/10/2021 • 4 minutes to read • [Edit Online](#)

Applies a filter to specified columns of a dataset

Category: [Data Transformation / Filter](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Apply Filter** module in Azure Machine Learning Studio (classic), to transform a column of values by applying a previously defined filter. Filters are used in digital signal processing to reduce noise or highlight a pattern. Thus, the values that you transform are always numeric, and typically represent some kind of audio or visual signal.

TIP

Are you looking for a different type of filter? Studio (classic) provides these modules for sampling data, getting a subset of data, removing bad values, or creating test and training sets: [Split Data](#), [Clean Missing Data](#), [Partition and Sample](#), [Apply SQL Transformation](#), [Clip Values](#). If you need to filter data as you read it from a source, see [Import Data](#). The options depend on the source type.

After determining which type of filter is best for your data source, you specify the parameters, and use **Apply Filter** to transform the dataset. Because the design of filters is separate from the process of applying a filter, filters are reusable. For example, if you frequently work with data used for forecasting, you might design several types of moving average filters to train and compare multiple models. You can also save the filter to apply to other experiments or to different datasets.

How to configure Apply Filter

1. Add the **Apply Filter** module to your experiment. You can find the IIR filter module under **Data Transformation**, in the **Filters** category.
2. To the right-hand input, connect a dataset that contains numeric values to one input.
3. To the left-hand input, connect an existing filter. You can re-use a saved filter, or you can configure a filter by using one of the following filter modules: [Threshold Filter](#), [Moving Average Filter](#), [Median Filter](#), [IIR Filter](#), [FIR Filter](#), [User-Defined Filter](#).
4. In the **Properties** pane of **Apply Filter**, click **Launch column selector** and choose the columns to which the filter should be applied.
5. Run the experiment, or right-click **Apply Filter** and click **Run selected**.

Results

The output contains only the data in the selected columns, transformed by applying the specified predefined

mathematical transformation.

If you want see other columns in the dataset, you can use the [Add Columns](#) module to merge the original and filtered datasets.

NOTE

The values in the original column have not been deleted or overwritten, and are still available in the experiment for reference. However, the output of the filter usually more useful for modeling.

Examples

For examples of how filters are used in machine learning, see the [Azure AI Gallery](#):

- [Filters](#): Demonstrates all filter types, using an engineered waveform dataset.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

- The **Apply Filter** module binds the specified type of filter to the selected columns. If you need to apply different types of filters to different columns, you should use [Select Columns in Dataset](#) to isolate the columns and apply different filter types in separate workflows. For more information, see [Select Columns in Dataset](#).
- The filters do not pass through data columns that are not affected by the filter. That is, the output of **Apply Filter** contains only the transformed numeric values. However, you can use the [Add Columns](#) module to join transformed values with the source dataset.

Filter periods

The filter period is determined in part by the filter type, as follows:

- For finite impulse response (FIR), simple moving average, and triangular moving average filters, the filter period is **finite**.
- For infinite impulse response (IIR), exponential moving average, and cumulative moving average filters, the filter period is **infinite**.
- For threshold filters, the filter period is always **1**.
- For median filters, regardless of the filter period, NaNs and missing values in the input signal do not produce new NaNs in output.

Missing values

This section describes the behavior when missing values are encountered, by filter type. In general, when a filter encounters a NaN or a missing value in the input dataset, the output dataset becomes spoiled with NaNs for some next number of samples, depending on the filter period. This has the following consequences:

- FIR, simple moving average, or triangular moving average filters have a finite period. As a result, any missing value will be followed by a number of NaNs equal to the filter order minus one.
- IIR, exponential moving average, or cumulative moving average filters have an infinite period. As a result, after the first missing value is encountered, NaNs will continue to propagate indefinitely.
- In a threshold filter, the period of a threshold filter is 1. As a result, missing values and NaNs do not propagate.
- For median filters, NaNs and missing values encountered in the input dataset do not produce new NaNs

in output, regardless of the filter period.

Expected inputs

NAME	TYPE	DESCRIPTION
Filter	IFilter interface	Filter implementation
Dataset	Data Table	Input dataset

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Column set	Any	ColumnSelection	NumericAll	Select the columns to filter

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Output dataset

See also

[Filter](#)

[A-Z Module List](#)

FIR Filter

3/10/2021 • 7 minutes to read • [Edit Online](#)

Creates a finite impulse response filter for signal processing

Category: [Data Transformation / Filter](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **FIR Filter** module in Azure Machine Learning Studio (classic), to define a kind of filter called a *finite impulse response* (FIR) filter. FIR filters have many applications in signal processing, and are most commonly used in applications that require a linear-phase response. For example, a filter might be applied to images used in healthcare to sharpen the overall image, eliminate noise, or focus on an imaged object.

NOTE

A filter is a transfer function that takes an input signal and creates an output signal based on the filter characteristics. For more general information about the user of filters in digital signal processing, see [Filter](#).

After you have defined a digital signal processing filter, you can apply the filter to data by connecting a dataset and the filter to the [Apply Filter](#) module. You can also save the filter for re-use with similar datasets.

TIP

Need to filter data from a dataset or remove missing values? Use these modules instead:

- [Clean Missing Data](#): Use this module to remove missing values or replace missing values with placeholders.
- [Partition and Sample](#): Use this module to divide or filter your dataset by criteria such as a range of dates, a specific value, or regular expressions.
- [Clip Values](#): Use this module to set a range and keep only the values within that range.

How to configure FIR Filter

1. Add the **FIR Filter** module to your experiment. You can find this module under **Data Transformation**, in the **Filter** category.
2. For **Order**, type an integer value that defines the number of active elements used to affect the filter's response. The *order* of the filter represents the length of the filter window.

For a FIR filter, the minimum order is 4.
3. For **Window**, select the shape of the data to which the filter will be applied. Azure Machine Learning supports the following types of windowing functions for use in finite impulse response filters:

Hamming: The *generalized Hamming window* provides a type of weighted averaging, which is commonly used in image processing and computer vision.

Blackman: A *Blackman window* applies a smoothly tapered curve function to the signal. The Blackman window has better stopband attenuation than other window types.

Rectangular: A *rectangular window* applies a consistent value inside the specified interval and applies no value elsewhere. The simplest rectangular window might replace n values in a data sequence with zeros, which makes it appear as though the signal suddenly turns on and off.

A rectangular window is also known as a *boxcar* or *Dirichlet window*.

Triangular: A triangular window applies filter coefficients in a step-wise fashion. The current value appears at the peak of the triangle, and then it declines with preceding or following values.

None: In some applications it is preferable not to use any windowing functions. For example, if the signal you are analyzing already represents a window or burst, applying a window function could deteriorate the signal-to-noise ratio.

4. For **Filter type**, select an option that defines how the filter is applied. You can specify that the filter exclude the target values, alter the values, reject the values, or pass them through.

Lowpass: "Low pass" means that the filter passes through lower values, and removes the higher values. For example, you might use this to remove high-frequency noise and data peaks from a signal.

This filter type has a smoothing effect on the data.

Highpass: "High pass" means that the filter passes through higher values, and removes lower values. You might use this to remove low frequency data, such as a bias or offset, from a signal.

This filter type preserves sudden changes and peaks in a signal.

Bandpass: "Band pass" means that it passes through the specified band of values, and remove others. You might use this filter to preserve the data from a signal with frequency characteristics at the intersection between the highpass and lowpass filters.

Bandpass filters are created by combining a highpass and a lowpass filter. The highpass filter cutoff frequency represents the lower cutoff, and the lowpass filter frequency represents the higher cutoff.

This filter type is good at removing a bias and smoothing a signal.

Bandstop: "Band stop" means that it blocks specified signals. In other words, it removes data from a signal with frequency characteristics that are rejected by the low pass and the highpass filters.

This filter type is good at preserving the signal bias and sudden changes.

5. Depending on the type of filter you chose, you must set one or more cutoff values.

Use the **High cutoff** and **Low cutoff** options to define an upper and/or a lower threshold for values. One or both of these options are required to specify which values are rejected or passed through. A **bandstop** or **bandpass** filter requires that you set both high and low cutoff values. Other filter types, such as the **lowpass** filter, require that you set only the low cutoff value.

6. Select the **Scale** option if scaling should be applied to coefficients; otherwise leave blank.

7. Connect the filter to [Apply Filter](#), and connect a dataset.

Use the column selector to specify which columns the filter should be applied to. By default, the [Apply Filter](#) module will use the filter for all selected numeric columns.

8. Run the experiment.

No computations are performed until you connect a dataset to the [Apply Filter](#) module and run the experiment. At that point, the specified transformation is applied to the selected numeric columns.

NOTE

The **FIR Filter** module does not provide the option to create an indicator column. Column values are always transformed in place.

Examples

For examples of how filters are used in machine learning, see this experiment in the [Azure AI Gallery](#):

- [Filters](#): This experiment demonstrates all filter types, using an engineered waveform dataset.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

FIR filters have these characteristics:

- FIR filters do not have feedback; that is, they use the previous filter outputs.
- FIR filters are more stable, because the impulse response will always return to 0.
- FIR filters require a higher order to achieve the same selectivity as infinite impulse response (IIR) filters.
- Like other filters, the FIR filter can be designed with a specific cutoff frequency that preserves or rejects frequencies that compose the signal.

Calculating coefficients over the filter window

The window type determines the trade-off between selectivity (width of the transition band in which frequencies are neither fully accepted nor rejected) and suppression (the total attenuation of frequencies to be rejected). The windowing function is applied to the ideal filter response to force the frequency response to zero outside of the window. Coefficients are selected by sampling the frequency response within the window.

The number of coefficients returned by the **FIR Filter** module is equal to the filter order plus one. The coefficient values are determined by filter parameters and by the windowing method, and are symmetric to guarantee a linear phase response

Scaling of coefficients

The **FIR Filter** module returns filter coefficients, or tap weights, for the created filter.

The coefficients are determined by the filter, based on the parameters you enter (such as the order). If you want to specify custom coefficients, use the [User-Defined Filter](#) module.

When **Scale** is set to **True**, filter coefficients will be normalized, such that the magnitude response of the filter at the center frequency of the passband is 0. The implementation of normalization in Azure Machine Learning Studio (classic) is the same as in the `fir1` function in MATLAB.

Typically, in the window design method, you design an ideal infinite impulse response (IIR) filter. The window function is applied to the waveform in the time domain, and multiplies the infinite impulse response by the window function. This results in the frequency response of the IIR filter being convolved with the frequency response of the window function. However, in the case of FIR filters, the input and filter coefficients (or tap weights) are convolved when applying the filter.

Selectivity and stop band attenuation

The following table compares selectivity with stop band attenuation for a FIR filter with length n by using

different windowing methods:

WINDOW TYPE	TRANSITION REGION	MINIMUM STOPBAND ATTENUATION
Rectangular	0.041n	21 dB
Triangle	0.11n	26 dB
Hann	0.12n	44 dB
Hamming	0.23n	53 dB
Blackman	0.2n	75 dB

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Order	>=4	Integer	5	Specify the filter order
Window	Any	WindowType		Specify the type of window to apply
Filter type	Any	FilterType	LowPass	Select the type of filter to create
Low cutoff	[double.Epsilon;9999999]	Float	0.3	Set the low cutoff frequency
High cutoff	[double.Epsilon;9999999]	Float	0.7	Set the high cutoff frequency
Scale	Any	Boolean	True	If true, filter coefficients will be normalized

Output

NAME	TYPE	DESCRIPTION
Filter	IFilter interface	Filter implementation

Exceptions

EXCEPTION	DESCRIPTION
NotInRangeValue	Exception occurs if parameter is not in range.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Filter](#)

[Apply Filter](#)

[A-Z Module List](#)

IIR Filter

3/10/2021 • 5 minutes to read • [Edit Online](#)

Creates an infinite impulse response filter for signal processing

Category: [Data Transformation / Filter](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **IIR Filter** module in Azure Machine Learning Studio (classic), to create an *infinite impulse response* (IIR) filter.

Filters are an important tool in digital signal processing, and are used to improve the results of image or voice recognition. In general, a filter is a transfer function that takes an input signal and creates an output signal based on the filter characteristics. For more general information about the user of filters in digital signal processing, see [Filter](#).

An **IIR filter** is a particular type of filter; typical uses of an IIR filter would be to simplify cyclical data that includes random noise over a steadily increasing or decreasing trend. The IIR filter you create with this module defines a set of constants (or coefficients) that alter the signal that is passed through. The word *infinite* in the name refers to the feedback between the outputs and the series values.

After you have defined a filter that meets your needs, you can apply the filter to data by connecting a dataset and the filter to the [Apply Filter](#) module.

TIP

A filter is a transfer function that takes an input signal and creates an output signal based on the filter characteristics. For more general information about the user of filters in digital signal processing, see [Filter](#).

After you have defined a filter that meets your needs, you can apply the filter to data by connecting a dataset and the filter to the [Apply Filter](#) module.

TIP

Need to filter data from a dataset or remove missing values? Use these modules instead:

- [Clean Missing Data](#): Use this module to remove missing values or replace missing values with placeholders.
- [Partition and Sample](#): Use this module to divide or filter your dataset by criteria such as a range of dates, a specific value, or regular expressions.
- [Clip Values](#): Use this module to set a range and keep only the values within that range.

How to configure IIR Filter

1. Add the **IIR Filter** module to your experiment. You can find this module under **Data Transformation**, in the **Filter** category.
2. For **Order**, type an integer value that defines the number of active elements used to affect the filter's response. The *order* of the filter represents the length of the filter window.

For an IIR filter, the minimum order is 4.

3. For **Filter kind**, choose the algorithm that is used to compute filter coefficients. The *filter kind* designates the mathematical transfer function that controls frequency response and frequency suppression. Azure Machine Learning supports these kinds of filters commonly used in digital signal processing:

Butterworth: A Butterworth filter is also called a *maximally flat magnitude filter* because it constrains the response (change in signal) in the passband and the stopband.

Chebyshev Type 1: Chebyshev filters are intended to minimize the errors between the idealized and the actual filter characteristics over the range of the filter. Type 1 Chebyshev filters leave more ripple in the passband.

Chebyshev Type 2: Type 2 Chebyshev filters have the same general characteristics as Type 1 Chebyshev filters, but they leave more ripple in the stopband.

4. For **Filter type**, select an option that defines how the filter will affect the values in the input signal. You can specify that the filter exclude values above or below a cutoff point, or specify that the filter either reject or pass through values in a specified frequency range.

LowPass: Allows low frequency values (below the cutoff value) to pass and attenuates other values.

HighPass: Allows high frequency values (above the cutoff value) to pass and attenuates other values.

Bandpass: Allows signals in the range that is specified by the low and high cutoff values to pass and attenuates other values.

BandStop: Allows signals outside the range specified by the low and high cutoff values to pass and attenuates values within the range.

5. Specify the high or low cutoff values, or both, as a value between 0 and 1, representing a normalized frequency. For **High cutoff**, type a value that represents the upper frequency boundary. For **Low cutoff**, type a value that represents the lower frequency boundary.

6. For **Ripple**, specify the amount of *ripple* to tolerate when you define your filter. Ripple refers to a small variation that occurs periodically. Ripple is usually considered an unwanted effect, but you can compensate for ripple by adjusting other filter parameters, such as the filter length. Not all filters produce ripple.

7. Add the [Apply Filter](#) module to your experiment, and connect the filter you designed, and the dataset containing the values you want to modify.

Use the column selector to specify which columns of the dataset to which the filter should be applied. By default, the [Apply Filter](#) module will use the filter for all selected numeric columns.

8. Run the experiment to apply the transformation.

NOTE

The IIR Filter module does not provide the option to create an indicator column. Column values are always transformed in place.

Examples

For examples of how filters are used in machine learning, see this experiment in the [Azure AI Gallery](#):

- [Filters](#): This experiment demonstrates all filter types, using an engineered waveform dataset.

Technical Notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

An IIR filter returns feed forward and feed backward coefficients, which are represented by way of a transfer function. Here is an example representation:

$$y[n] = \frac{1}{a_0} \left(\sum_{i=0}^N b_i x[n-i] - \sum_{i=0}^N a_i y[n-i] \right)$$

Where:

- N : filter order
- b_i : feed forward filter coefficients
- a_i : feed backward filter coefficients
- $x[n]$: the input signal
- $y[n]$: the output signal

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Order	[4;13]	Integer	5	Specify the filter order
Filter kind	Any	IIRFilterKind		Select the kind of IIR filter to create
Filter type	Any	FilterType		Select the filter band type
Low cutoff	[double.Epsilon;9999 999]	Float	0.3	Set the low cutoff value
High cutoff	[double.Epsilon;9999 999]	Float	0.7	Set the high cutoff value
Ripple	>=0.0	Float	0.5	Specify the amount of ripple in the filter

Output

NAME	TYPE	DESCRIPTION
Filter	IFilter interface	Filter implementation

Exceptions

EXCEPTION	DESCRIPTION
NotInRangeValue	Exception occurs if parameter is not in range.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Filter](#)

[Apply Filter](#)

[A-Z Module List](#)

Median Filter

3/10/2021 • 3 minutes to read • [Edit Online](#)

Creates a median filter used to smooth data for trend analysis

Category: [Data Transformation / Filter](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Median Filter** module in Azure Machine Learning Studio (classic), to define a *median filter* for applying to a series of values that represent a digital input signal or image.

Median filters are widely used in image recognition to reduce noise so that features can more easily be detected.

NOTE

A filter is a transfer function that takes an input signal and creates an output signal based on the filter characteristics. In digital signal processing, the use of filters can improve the results of image or voice recognition. For more information, see [Filter](#).

After you have defined a filter transformation that meets your needs by using the **Median Filter** module, you can apply the filter to data by connecting a dataset and the filter to the [Apply Filter](#) module.

TIP

Need to filter data from a dataset or remove missing values? Use these modules instead:

- [Clean Missing Data](#): Use this module to remove missing values or replace missing values with placeholders.
- [Partition and Sample](#): Use this module to divide or filter your dataset by criteria such as a range of dates, a specific value, or regular expressions.
- [Clip Values](#): Use this module to set a range and keep only the values within that range.

How to configure Median Filter

1. Add the **Median Filter** to your experiment. You can find this module under **Data Transformation**, in the **Filter** category.
2. For **Length**, type an integer value that defines the total size of the window across which the filter is applied. This is also called the filter *mask*.

The value should be an odd, positive-valued integer. If you specify an even number, the mask size is reduced by one.

By default the mask begins at the current value and creates a window centered on the current value.

For example, if you type 5 as the **Length** or window size, the median value is computed across a sliding window consisting of 5 values centered on the current value. If you type 4, the mask is reduced to 3 values, centered on the index value.

3. Connect the filter to [Apply Filter](#), and connect a dataset.

Use the column selector to specify which columns of the dataset to which the filter should be applied. By default, the [Apply Filter](#) module will use the filter for all selected numeric columns.

4. Run the experiment. The following operations are applied to the selected columns:

- For each set of values included in the window or mask, the filter algorithm computes the median.
- The current (or index) value is replaced with the median value.

Examples

For examples of how filters are used in machine learning, see this experiment in the [Azure AI Gallery](#):

- [Filters](#): This experiment demonstrates all filter types, using an engineered waveform dataset.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

Each entry in the output signal is equal to the median of the entries in a subset (mask) of the input signal, and centered at the corresponding index. The mask size should be an odd, positive-valued integer.

If you provide this method with an even-valued mask size, it is reduced by one. For example, given `m=2q+1`, the filter is defined as: `yi = median[{xi-q, ..., xi+q}]`

Values beyond the borders of the input signal are assumed to equal the value at the border. That is, if n is the length of the input signal:

$$x_i = \begin{cases} x_1, & \forall i < 1 \\ x_n, & \forall i > n \end{cases}$$

For more information about median filters, this Wikipedia article provides a good explanation of the theory and application:

[Wikipedia: Median Filters](#)

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Length	≥ 1	Integer	5	Length of the filter window

Output

NAME	TYPE	DESCRIPTION
Filter	IFilter interface	Filter implementation

See also

[Filter](#)

[Apply Filter](#)

[A-Z Module List](#)

Moving Average Filter

3/10/2021 • 4 minutes to read • [Edit Online](#)

Creates a moving average filter used to smooth data for trend analysis

Category: [Data Transformation / Filter](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Moving Average Filter** module in Azure Machine Learning Studio (classic), to calculate a series of one-sided or two-sided averages over a dataset, using a window length that you specify.

After you have defined a filter that meets your needs, you can apply it to selected columns in a dataset by connecting it to the [Apply Filter](#) module. The module does all the calculations and replaces values within numerical columns with corresponding moving averages.

You can use the resulting moving average for plotting and visualization, as a new smooth baseline for modeling, for calculating variances against calculations for similar periods, and so on.

TIP

Need to filter data from a dataset or remove missing values? Use these modules instead:

- [Clean Missing Data](#): Use this module to remove missing values or replace missing values with placeholders.
- [Partition and Sample](#): Use this module to divide or filter your dataset by criteria such as a range of dates, a specific value, or regular expressions.
- [Clip Values](#): Use this module to set a range and keep only the values within that range.

Understanding and using moving averages

This type of average helps you reveal and forecast useful temporal patterns in retrospective and real-time data. The simplest type of moving average starts at some sample of the series, and uses the average of that position plus the previous n positions instead of the actual value. (You can define n as you like.) The longer the period n across which the average is computed, the less variance you will have among values. Also, as you increase the number of values used, the less effect any single value has on the resulting average.

A moving average can be *one-sided* or *two-sided*. In a one-sided average, only values preceding the index value are used. In a two-sided average, past and future values are used.

For scenarios in which you are reading streaming data, cumulative and weighted moving averages are particularly useful. A *cumulative moving average* takes into account the points preceding the current period.

You can weight all data points equally when computing the average, or you can ensure that values nearer the current data point are weighted more strongly. In a *weighted moving average*, all weights must sum to 1.

In an *exponential moving average*, the averages consist of a *head* and a *tail*, which can be weighted. A lightly weighted tail means that the tail follows the head quite closely, so the average behaves like a moving average on a short weighting period. When tail weights are heavier, the average behaves more like a longer simple moving average.

How to configure Moving Average Filter

1. Add the **Moving Average Filter** module to your experiment. You can find this module under **Data Transformation**, in the **Filter** category.
2. For **Length**, type a positive whole number value that defines the total size of the window across which the filter is applied. This is also called the filter *mask*. For a moving average, the length of the filter determines how many values are averaged in the sliding window.

Longer filters are also called *higher order* filters, and provide a larger window of calculation and a closer approximation of the trend line.

Shorter or *lower order* filters use a smaller window of calculation and more closely resemble the original data.

3. For **Type**, choose the type of moving average to apply.

Azure Machine Learning Studio (classic) supports the following types of moving average calculations:

Simple: A simple moving average (SMA) is calculated as an unweighted rolling mean.

Triangular: Triangular moving averages (TMA) are averaged twice for a smoother trend line. The word triangular is derived from the shape of the weights that are applied to the data, which emphasizes central values.

Exponential Simple: An exponential moving average (EMA) gives more weight to the most recent data. The weighting drops off exponentially.

Exponential: A modified exponential moving average calculates a running moving average, where calculating the moving average at any one point considers the previously computed moving average at all preceding points. This method yields a smoother trend line.

Cumulative: Given a single point and a current moving average, the cumulative moving average (CMA) calculates the moving average at the current point.

4. Add the dataset that has the values you want to compute a moving average for, and add the [Apply Filter](#) module.

Connect the **Moving Average Filter** to the left-hand input of [Apply Filter](#), and connect the dataset to the right-hand input.

5. In the [Apply Filter](#) module, use the column selector to specify which columns the filter should be applied to. By default, the filter transformation is applied to all numeric columns, so be sure to exclude any columns that don't have appropriate data.
6. Run the experiment.

For each set of values defined by the filter length parameter, the current (or index) value is replaced with the moving average value.

Examples

For examples of how filters are used in machine learning, see this experiment in the [Azure AI Gallery](#):

- [Filters](#): This experiment demonstrates all filter types, using an engineered waveform dataset.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Length	≥ 1	Integer	5	Set the length of the moving average window
Type	Any	MovingAverageType		Specify the type of moving average to create

Outputs

NAME	TYPE	DESCRIPTION
Filter	IFilter interface	Filter implementation

See also

[Filter](#)
[Apply Filter](#)
[A-Z Module List](#)
[Additional filter samples](#)

Threshold Filter

3/10/2021 • 6 minutes to read • [Edit Online](#)

Creates a threshold filter that constrains values

Category: [Data Transformation / Filter](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Threshold Filter** module in Azure Machine Learning Studio (classic), to define a filter that restricts numeric values to a specified range.

Threshold filters are commonly used in digital signal processing. A threshold filter examines each value of the input dataset and changes all values that do not meet the boundary conditions. You typically would use this type of filter for the following applications:

- Replace all negatively signed measurements with a value of zero.
- Convert a gray-scale image to black and white areas by defining a numerical boundary value for all pixels.

After you have defined a filter that meets your needs, you can apply the filter to data by connecting a dataset and the filter to the [Apply Filter](#) module.

The output of the [Apply Filter](#) module is a dataset containing the selected columns, transformed as specified by the **Threshold Filter** settings.

Alternatively, if you select the **Indicator** option, instead of returning the filter values, a column is returned containing Boolean values that indicates whether the value in each row met the specified filter condition or not. This can be useful when you are testing a new filter.

TIP

Need to filter data from a dataset or remove missing values? Use these modules instead:

- [Clean Missing Data](#): Use this module to remove missing values or replace missing values with placeholders.
- [Partition and Sample](#): Use this module to divide or filter your dataset by criteria such as a range of dates, a specific value, or regular expressions.
- [Clip Values](#): Use this module to set a range and keep only the values within that range.

How to configure Threshold Filter

1. Add the **Threshold Filter** module to your experiment. You can find this module under **Data Transformation**, in the **Filter** category.
2. For **Type**, specify the type of filter to apply:

- **LessThan**: Changes values that are less than the specified level to the boundary level, and passes through all other values.
- **GreaterThan**: Changes values that are greater than the specified level to the boundary level, and passes through all other values.
- **MagnitudeLessThan**: Changes values less than the specified level to the boundary level but preserves the sign of the original value.
- **MagnitudeGreaterThan**: Changes values greater than the specified level to the boundary level but preserves the sign of the original value.
- **InRange**: Passes through all values that fall within the specified range, and changes values outside the range to the closest boundary value.
- **OutOfRange**: Passes through all values that fall outside the specified range, and changes values inside the range to the closest boundary value.
- **InRangeWithStd**: Passes through all values that fall within the specified range of standard deviations, and changes values outside the range to the closest boundary value.
- **OutOfRangeWithStd**: Passes through all values that fall outside the specified range of standard deviations, and changes values inside the range to the closest boundary value.

3. For **Level**, type the boundary value to apply in each type of threshold.

- If you select the **LessThan** filter, the number you specify defines the lowest value that can be passed through without replacement.
- If you select the **GreaterThan** filter, the number you specify defines the greatest value that can be passed through without replacement.
- If you select the **MagnitudeLessThan** filter, type a single positive or negative number for **Level**. Any value that is less than that value is replaced with the level value.
- If you select the **MagnitudeGreaterThan** filter, type a single positive or negative number for **Level**. Any value that is greater than that value is replaced with the level value.
- If you select the filters, **InRange** or **OutOfRange**, specify the upper or lower bounds. For **Lower boundary**, type the lowest number to include in the range. For **Upper boundary**, type the highest number to include in the range.
- If you chose one of the filter types that uses standard deviations (**InRangeWithStd**, **OutOfRangeWithStd**), you must specify the **Alpha** constant. The values of alpha times the deviation is used to calculate the filter result.

4. Optionally, select the **Indicator** option to generate a column that only indicates whether the value would be affected by the filter. If you leave **Indicator** unselected, the filter generates the replacement values.

5. Connect the filter to [Apply Filter](#), and connect a dataset.

Use the column selector to specify which columns the filter should be applied to. By default, the [Apply Filter](#) module applies the filter transformation to all selected numeric columns.

6. Run the experiment.

No computations are performed until you connect a dataset to the [Apply Filter](#) module and run the experiment. At that point, the specified transformation is applied to the selected numeric columns.

Examples

For examples of how filters are used in machine learning, see this experiment in the [Azure AI Gallery](#):

- **Filters:** This experiment demonstrates all filter types, using an engineered waveform dataset.

Examples of indicator values

The following example assumes that you apply a threshold filter that specifies a range with a lower boundary of 2 and an upper boundary of 4:

VALUE	INDICATOR	REPLACE WITH
1	FALSE	2
2	TRUE	2
3	TRUE	3
4	TRUE	4
5	FALSE	4

Examples of magnitude in a filter

The filter types **MagnitudeLessThan** and **MagnitudeGreaterthan** first evaluate the value against the specified level, and then provide a replacement value that varies depending on the sign of the original values.

Examples of magnitude filters

The filter types **MagnitudeLessThan** and **MagnitudeGreaterthan** first evaluate the value against the specified level, and then provide a replacement value that varies depending on the sign of the original values.

For example, the following table shows the results when using a **MagnitudeLessThan** filter with values of 5 and -5.

SOURCE VALUE	LEVEL	NEW VALUE
3.07	5	5 Value is less than 5; therefore value is replaced with Level
3.07	-5	3.07 Value is not less than -5; therefore value is not replaced
-3.93	5	-5 Value is less than 5; therefore value is replaced with Level but sign of original value is preserved
-3.93	-5	-3.93 Value is not less than -5; therefore value is not replaced

SOURCE VALUE	LEVEL	NEW VALUE
5.75	5	5.75 Value is not less than -5; therefore value is not replaced
-5.75	-5	-5.75 Value is not less than -5; therefore value is not replaced

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

The Threshold Filter module uses the following methods to define threshold values, depending on the filter type:

- **LessThan**: The less-than mode is defined as:

$$x_i = \begin{cases} level, & x < level \\ x, & otherwise \end{cases}$$

MagnitudeLessThan: The less-than-magnitude mode is defined as:

$$y = \begin{cases} level, & |x| < level, x \geq 0 \\ -level, & |x| < level, x < 0 \\ x, & otherwise \end{cases}$$

For complex inputs, the magnitude of each element is restricted as shown by this formula:

$$y = \begin{cases} \frac{x-level}{|x|}, & |x| > level \\ x, & otherwise \end{cases}$$

- **MagnitudeGreaterThan**: The greater than magnitude mode is defined as:

$$y = \begin{cases} level, & |x| > level, x \geq 0 \\ -level, & |x| > level, x < 0 \\ x, & otherwise \end{cases}$$

For complex inputs, the magnitude of each element is restricted as shown by this formula:

$$y = \begin{cases} \frac{x-level}{|x|}, & |x| > level \\ x, & otherwise \end{cases}$$

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Type	Any	ThresholdType	LessThan	Select the threshold method to use
Indicator	Any	Boolean	false	Select this option to return a column that contains a True/False indication of whether the value met the filter condition, rather than the filtered values.
Level	Any	Float	0.0	Set the replacement value
Lower boundary	Any	Float	-1.0	Specify the lower boundary of the range
Upper boundary	Any	Float	1.0	Specify the upper boundary of the range
Alpha	Any	Float	3.0	Use this value, multiplied by the calculated standard deviation, as the threshold

Output

NAME	TYPE	DESCRIPTION
Filter	IFilter interface	Filter implementation

See also

[Filter](#)

[Apply Filter](#)

[A-Z Module List](#)

User-Defined Filter

3/10/2021 • 4 minutes to read • [Edit Online](#)

Creates a custom finite or infinite impulse response filter

Category: [Data Transformation / Filter](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **User-Defined Filter** module in Azure Machine Learning Studio (classic), to define a custom filter by using a finite impulse response (FIR) filter or an infinite impulse response (IIR) filter with coefficients that you specify.

A filter is a transfer function that takes an input signal and creates an output signal based on the filter characteristics. For more general information about the user of filters in digital signal processing, see [Filter](#). This module is particularly useful for applying a set of previously derived filter coefficients to your data.

After you have defined a filter that meets your needs, you can apply the filter to data by connecting a dataset and the filter to the [Apply Filter](#) module.

TIP

Need to filter data from a dataset or remove missing values? Use these modules instead:

- [Clean Missing Data](#): Use this module to remove missing values or replace missing values with placeholders.
- [Partition and Sample](#): Use this module to divide or filter your dataset by criteria such as a range of dates, a specific value, or regular expressions.
- [Clip Values](#): Use this module to set a range and keep only the values within that range.

How to configure User-Defined Filter

1. Add the **User-Defined Filter** module to your experiment in Studio (classic). You can find this module under **Data Transformation**, in the **Filter** category.
2. In the **Properties** pane, choose a type of filter: FIR filter, or IIR filter.
3. Provide the coefficients to apply in the filter. The requirements for the coefficients differ depending on whether you choose a FIR filter or an IIR filter.
 - For a FIR filter, you specify a vector of feed-forward coefficients. The length of the vector determines the filter's order. A FIR filter is effectively a moving average, so the configuration values apply a moving average to filter a data sequence.
 - For an IIR filter, you apply custom feed-forward and feed-backward coefficients. See the [Examples](#) section for some tips.

4. Connect the filter to [Apply Filter](#), and connect a dataset.

Use the column selector to specify which columns of the dataset to which the filter should be applied. By default, the [Apply Filter](#) module will use the filter for all selected numeric columns.

5. Run the experiment.

The specified transformations are applied to the selected numeric columns only when you run the experiment using [Apply Filter](#).

Examples

For more examples of how filters are used in machine learning, see the [Azure AI Gallery](#):

- [Filters](#): Demonstrates all filter types. The example uses an engineered waveform dataset to more easily illustrate the effects of the different filters.

FIR filter example: Exponential weighted moving average

For an exponentially weighted moving average, all coefficients are less than one and the sum of all coefficients equals one. Therefore, the variance of the weighted average will always be less than the input values.

For example, for a FIR filter to approximate an exponentially weighted moving average (WMA), you would supply a comma-separated list of coefficients for the value for the feed-forward parameter:

```
0.01818182, 0.03636364, 0.05454545, 0.07272727, 0.09090909, 0.10909091, 0.12727273, 0.14545455, 0.16363636,  
0.18181818
```

FIR filter example: Exponential weighted moving average (Deslauriers-Dubuc interpolation)

This FIR filter approximates a triangularly weighted moving average (WMA). You define the coefficients by supplying a comma-separated series of values for the feed-forward parameters, such as these:

```
0.0625, 0.0625, 0.2500, 0.3750, 0.2500, 0.0625
```

The values used in this custom FIR filter represent a vector of feed-forward coefficients obtained by using the Deslauriers-Dubuc method of finite sequencing. For more information, see [Dubuc-Deslauriers Subdivision for Finite Sequences and Interpolation Wavelets on an Interval](#).

IIR filter example: Notch filter

A good example of an application for a user-defined IIR filter is to define a *notch filter*, also called a *bandstop filter*. The desired notch filter attenuates a -3dB rejection band, *fb*, centered at a notch frequency, *fn*, with a sampling frequency, *fs*.

In this case, the digital notch filter can be represented by the following formula:

$$G(z) = \frac{(1 + a_2) - 2a_1z^{-1} + (1 + a_2)z^{-2}}{1 - a_1z^{-1} + a_2z^{-2}}$$

This formula assumes:

$$a_2 = \frac{1 - \tan(\Omega T/2)}{1 + \tan(\Omega T/2)}$$

From this formula, we can get the feed-forward coefficient:

$$1 + a_2, -2a_1, 1 + a_2$$

The feed-backward coefficients would be as follows:

$$1, -a_1, a_2$$

Example of IIR Filter: Notch filter 2

The following example shows a notch filter with a notch frequency of $f_n = 1250 \text{ Hz}$ and a -3 dB rejection band of $f_b = 100 \text{ Hz}$, with sampling frequency of $f_s = 10 \text{ kHz}$.

$$\omega_0 T = 2\pi f_n / f_s$$

Using the following formula, you get $a_2 = 0.93906244$ and $a_1 = 1.3711242$:

$$\Omega T = 2\pi f_b / f_s$$

From this, you can get the following feed-forward (b) and feed-backward (a) coefficients:

$$b = [1.9390624, -2.7422484, 1.9390624]$$

$$a = [1, -1.3711242, 0.9390624]$$

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Type of filter	any	ImpulseResponse		Specify the type of filter to customize
Forward	any	String	"1.0"	Type a series of feed-forward coefficients
Backward	any	String	"1.0"	Type a series of feed-backward filter coefficients

Output

NAME	TYPE	DESCRIPTION
Filter	IFilter interface	Filter implementation

Exceptions

EXCEPTION	DESCRIPTION
ParameterParsing	An exception occurs if one or more parameters could not be parsed or converted from the specified type into the type that is required by the target method.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Apply Filter](#)

[A-Z Module List](#)

Data Transformation - Learning with Counts

3/10/2021 • 5 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that support count-based featurization.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Learning with counts is an efficient way to create a compact set of dataset features that are based on counts of the values. You can use the modules in this category to build a set of counts and features. Later, you can update the counts and the features to take advantage of new data, or merge two sets of count data.

About count-based featurization

The basic idea of *count-based featurization* is that by calculating counts, you can quickly and easily get a summary of what columns contain the most important information. The module counts the number of times a value appears, and then provides that information as a feature for input to a model.

Imagine that you're validating a credit card transaction. A crucial piece of information is where this transaction came from. One of the most common encodings of the transaction origin is the postal code. However, there might be as many as 40,000 postal codes, zip codes, and geographical codes to account for. Does your model have the capacity to learn 40,000 more parameters? If you give it that capacity, do you have enough training data to prevent it from overfitting?

If you have good data, with lots of samples, such fine-grained local granularity can be powerful. However, if you have only one sample of a fraudulent transaction, from a small locality, does it mean that all of the transactions from that place are bad, or that you don't have enough data?

One solution is to learn with counts. Instead of introducing 40,000 more features, you can observe the counts and proportions of fraud for each postal code. By using these counts as features, you get information about the strength of the evidence for each value. Moreover, by encoding the relevant statistics of the counts, the learner can use the statistics to decide when to change their approach and instead use other features to get the information.

Count-based learning is attractive for many reasons. With count-based learning, you have fewer features, which requires fewer parameters. Fewer parameters makes for faster learning, faster prediction, smaller predictors, and less potential to overfit.

How count-based features are created

A basic example can help demonstrate how count-based features are created and applied. Suppose you have the following table like this, with labels and inputs. Each case (or row or sample) has a set of values in columns. In this example, the values are A and B.

LABEL COLUMN	INPUT VALUE
0	A

LABEL COLUMN	INPUT VALUE
0	A
1	A
0	B
1	B
1	B
1	B

These are the steps you take to create count-based features:

1. For a specific set of values, find all the other cases in that dataset that have the same value. In this case, there are three instances of A and four instances of B.
2. Count the class membership of each value as a feature in itself. In this case, you get a small matrix: there are two cases where A = 0; one case where A = 1; one case where B = 0; and three cases where B = 1.
3. Based on this matrix, you get a variety of count-based features. These include a calculation of the log-odds ratio and the counts for each target class. The table in the next section displays the data.

Sample table of count-based features

LABEL	0_0_CLASS000_COUNT	0_0_CLASS001_COUNT	0_0_CLASS000_LOGOdds	0_0_ISBACKOFF
0	2	1	0.510826	0
0	2	1	0.510826	0
1	2	1	0.510826	0
0	1	3	-0.8473	0
1	1	3	-0.8473	0
1	1	3	-0.8473	0
1	1	3	-0.8473	0

Examples

In [Use Azure Machine Learning to build clickthrough prediction models](#), the Microsoft Machine Learning team provides a detailed walkthrough of how to use counts in machine learning. The article compares the efficacy of count-based modeling with other methods.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

How the log-loss value is calculated

The log-loss value is not the plain log-odds. In this case, the prior distribution is used to smooth the log-odds

computation.

Suppose you have a dataset that's used for binary classification. In this dataset, the prior frequency for class 0 is p_0 , and the prior frequency for class 1 is $p_1 = 1 - p_0$. For a specific training example feature, the count for class 0 is x_0 , and the count for class 1 is x_1 .

Under these assumptions, the log-odds is computed as $\text{LogOdds} = \text{Log}(x_0 + c * p_0) - \text{Log}(x_1 + c * p_1)$, where c is the prior coefficient, which can be set by the user. The log function uses the natural base.

In other words, for each class i :

```
Log_odds[i] = Log( (count[i] + prior_coefficient * prior_frequency[i]) / (sum_of_counts - count[i]) + prior_coefficient \* (1 - prior_frequency[i]))
```

If the prior coefficient is positive, the log-odds can be different from

```
Log(count[i] / (sum_of_counts - count[i])).
```

Why the log-odds aren't computed for some items

By default, all items with a count that's less than 10 are collected in a single bucket called the "garbage bin." You can change this value by using the **Garbage bin threshold** option in the [Modify Count Table Parameters](#) module.

List of modules

The **Learning with Counts** category includes the following modules:

- [Build Counting Transform](#): Creates a count table and count-based features from a dataset, and then saves the table and features as a transformation.
- [Export Count Table](#): Exports a count table from a counting transform. This module supports backward compatibility with experiments that create count-based features by using Build Count Table (deprecated) and Count Featurizer (deprecated).
- [Import Count Table](#): Imports an existing count table. This module supports backward compatibility with experiments that create count-based features by using Build Count Table (deprecated) and Count Featurizer (deprecated). The module supports conversion of count tables to count transformations.
- [Merge Count Transform](#): Merges two sets of count-based features.
- [Modify Count Table Parameters](#): Modifies count-based features that are derived from an existing count table.

See also

- [Data Transformation](#)
- [Feature Selection](#)

Build Counting Transform

3/10/2021 • 12 minutes to read • [Edit Online](#)

Creates a transformation that turns count tables into features, so that you can apply the transformation to multiple datasets

Category: [Learning with Counts](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Build Counting Transform** module in Azure Machine Learning Studio (classic), to analyze training data. From this data, the module builds a *count table* as well as a set of *count-based features* that can be used in a predictive model.

A count table contains the joint distribution of all feature columns, given a specified label column. Such statistics are useful in determining which columns have the most information value. *Count-based featurization* is useful because such features are more compact than the original training data, but capture all the most useful information. You can use the module parameters to customize how the counts are transformed into the new set of count-based features.

After generating counts and transforming them into features, you can save the process as a transformation for re-use on related data. You can also modify the set of features without having to generate a new set of counts, or merge the counts and features with another set of counts and features.

The ability to re-use and re-apply count-based features is useful in scenarios such as these:

- New data becomes available to improve the coverage or balance of your dataset.
- Your original counts and features were based on a very large dataset that you don't want to re-process. By merging the counts you can update with new data.
- You want to ensure that the same set of count-based features is applied to all datasets that you are using in your experiment.

How to configure Build Counting Transform

You can create a count-based feature transformation directly from a dataset, and re-run it each time you run an experiment. Or, you can generate a set of counts, and then merge it with new data to create an updated count table.

- [Create count-based features from a dataset](#)

Start here if you have not created counts before. You use the **Build Counting Transform** module to create count tables and automatically generate a set of features.

This process creates a feature transformation that you can apply to a dataset, using the [Apply Transformation](#) module.

- [Merge counts and features from multiple datasets](#)

If you have already generated a count table from a previous dataset, generate counts on just the new data, or import an existing count table created in an earlier version of Azure Machine Learning. Then, merge the two sets of count tables.

This process creates a new feature transformation that you can apply to a dataset, using the [Apply Transformation](#) module.

Create count-based features from a dataset

1. In Azure Machine Learning Studio (classic), add the **Build Counting Transform** module to your experiment. You can find the module under **Data Transformation**, in the category **Learning with Counts**.
2. Connect the dataset you want to use as the basis for our count-based features.
3. Use the **Number of classes** option to specify the number of values in your label column.
 - For any binary classification problem, type .
 - For a classification problem with more than two possible outputs, you must specify in advance the exact number of classes to count. If you enter a number that is less than the actual number of classes, the module will return an error.
 - If your dataset contains multiple class values and the class label values are non-sequential, you must use [Edit Metadata](#) to specify that the column contains categorical values.
4. For the option, **The bits of hash function**, indicate how many bits to use when hashing the values.

It is generally safe to accept the defaults, unless you know that there are many values to count and a higher bit count might be needed.

5. In **The seed of hash function**, you can optionally specify a value to seed the hashing function. Setting a seed manually is typically done when you want to ensure that hashing results are deterministic across runs of the same experiment.
6. Use the **Module type** option to indicate the type of data that you will be counting, based on the storage mode:
 - **Dataset:** Choose this option if you are counting data that is saved as a dataset in Azure Machine Learning Studio (classic).
 - **Blob:** Choose this option if your source data used to build counts is stored as a block blob in Windows Azure storage.
 - **MapReduce:** Choose this option if you want to call Map/Reduce functions to process the data.

To use this option, the new data must be provided as a blob in Windows Azure storage, and you must have access to a deployed HDInsight cluster. When you run the experiment, a Map/Reduce job is launched in the cluster to perform the counting.

For very large datasets, we recommend that you use this option whenever possible. Although you might incur additional costs for using the HDInsight service, computation over large datasets might be faster in HDInsight.

For more information, see <https://azure.microsoft.com/services/hdinsight/>.

7. After specifying the data storage mode, provide any additional connection information for the data that is required:
 - If you are using data from Hadoop or blob storage, provide the cluster location and credentials.
 - If you previously used a [Import Data](#) module in the experiment to access data, you must re-enter the

account name and your credentials. The **Build Counting Transform** module accesses the data storage separately in order to read the data and build the required tables.

8. For **Label column or index**, select one column as the label column.

A label column is required. The column must already be marked as a label or an error is raised.

9. Use the option, **Select columns to count**, and select the columns for which to generate counts.

In general, the best candidates are high-dimensional columns, together with any other columns that are correlated with those columns.

10. Use the **Count table type** option to specify the format used for storing the count table.

- **Dictionary:** Creates a dictionary count table. All column values in the selected columns are treated as strings, and are hashed using a bit array of up to 31 bits in size. Therefore, all column values are represented by a non-negative 32-bit integer.

In general, you should use this option for smaller data sets (less than 1 GB), and use the **CMSketch** option for larger datasets.

After selecting this option, configure the number of bits used by the hashing function, and set a seed for initializing the hash function.

- **CMSketch:** Creates a *count minimum sketch table*. With this option, multiple independent hash functions with a smaller range are used to improve memory efficiency and reduce the chance of hash collisions. The parameters for hashing bit size and hashing seed have no effect on this option.

11. Run the experiment.

The module creates a *featurization transform* that you can use as input to the [Apply Transformation](#) module. The output of the [Apply Transformation](#) module is a transformed dataset that can be used to train a model.

Optionally, you can save the transform if you want to merge the set of count-based features with another set of count-based features. For more information, see [Merge Count Transform](#).

Merge counts and features from multiple datasets

1. In Azure Machine Learning Studio (classic), add the **Build Counting Transform** module to your experiment, and connect the dataset that contains the new data you want to add.
2. Use the **Module type** option to indicate the source of the new data. You can merge data from different sources.
 - **Dataset:** Choose this option if the new data is provided as a dataset in Azure Machine Learning Studio (classic).
 - **Blob:** Choose this option if the new data is provided as a block blob in Windows Azure storage.
 - **MapReduce:** Choose this option if you want to call Map/Reduce functions to process the data.

To use this option, the new data must be provided as a blob in Windows Azure storage, and you must have access to a deployed HDInsight cluster. When you run the experiment, a Map/Reduce job will be launched in the cluster to perform the counting.

For more information, see <https://azure.microsoft.com/services/hdinsight/>
3. After specifying the data storage mode, provide any additional connection information for the new data:
 - If you are using data from Hadoop or blob storage, provide the cluster location and credentials.
 - If you previously used a [Import Data](#) module in the experiment to access data, you must re-enter

the account name and your credentials. The reason is that the **Build Counting Transform** module accesses the data storage separately in order to read the data and build the required tables.

- When merging counts, the following options must be exactly the same in both counts tables:

- Number of classes
- The bits of hash function
- The seed of hash function
- Select columns to count

The label column can be different, as long as it contains the same number of classes.

- Use the **Count table type** option to specify the format and destination for the updated count table.

TIP

The format of the two count tables that you intend to merge must be the same. In other words, if you saved an earlier count table using the **Dictionary** format you cannot merge it with counts saved using the **CMSketch** format.

- Run the experiment.

The module creates a *featurization transform* that you can use as input to the [Apply Transformation](#) module. The output of the [Apply Transformation](#) module is a transformed dataset that can be used to train a model.

- To merge this set of counts with an existing set of count-based features, see [Merge Count Transform](#).

Examples

See these articles for more information about the counts algorithm and the efficacy of count-based modeling compared to other methods.

- [Using Azure ML to Build Click-through Prediction Models](#)
- [Big Learning Made Easy with Counts!](#)

The following experiments in the [Azure AI Gallery](#) demonstrate how to use count-based learning to build various predictive models:

- [Learning With Counts - Binary Classification](#)
- [Learning with Counts: Multiclass classification with NYC taxi data](#)
- [Learning with Counts: Binary classification with NYC taxi data](#)

Module parameters

The following parameters are used with all options:

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Number of classes	Integer	>=2	Required	2	The number of classes for the label.

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
The bits of hash function	Integer	[12;31]	Required	20	The number of bits of the range of hash function.
The seed of hash function	Integer	any	Required	1	The seed for the hash function.
Module type			Required	Dataset	The type of module to use when generating the count table.
Count table type	CountTableType	select from list	Required	Dictionary	Specify the format of the count table.

The following options apply when selecting the **blob** option.

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Blob name	String	any	Required		The name of the input blob. Do not include container name.
Account name	String	any	Required		The name of the storage account.
Account key	SecureString	any	Required		The key of the storage account.
Container name	String	any	Required		The Azure blob container that contains the input blob.
Count columns	String	any	Required		The one-based indexes of groups of columns to perform counting.
Label column	Integer	>= 1	Required	1	The one-based index of the label column.
Blob format		any	Required	CSV	The blob text file format.

The following parameters apply when using **MapReduce** to generate counts:

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Default storage account name	String	any	Required	none	The name of the storage account containing the input blob.
Default storage account key	SecureString	any	Required	none	The key of the storage account containing the input blob.
Default container name	String	any	Required	none	The name of the blob container to write the count table.
Cluster URI	String	any	Required	none	The URI to the HDInsight Hadoop cluster.
Username	String	any	Required	none	The username to login to the HDInsight Hadoop cluster.

The following parameters define the format of the count table:

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Count table type	CountTableType	List	Required	Dictionary	Type of the count table.
Label column index or name	ColumnSelection		Required if count table saved as Dataset	none	Select the label column.
Select columns to count	ColumnSelection		Required if count table saved as Dataset		Select columns for counting. These columns are considered as categorical features.
Depth of CM sketch table	Integer	>=1	Required if count table uses CMSketch format	4	The depth of the CM sketch table, which equals the number of hash functions.
Width of CM sketch table	Integer	[1;31]	Required if count table uses CMSketch format	20	The width of the CM sketch table, which is the number of bits of the range of hash function.

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Label column index or namecolumn	ColumnSelection		Required if count table saved as Dataset		Selects the label column.
Select columns to count	ColumnSelection		Required if count table saved as Dataset		Selects columns for counting. These columns are considered as categorical features.
Count table type			Required if count table saved as Dataset	Dictionary	Specifies the type of the count table.
Depth of CM sketch table	Integer	>=1	Required if count table saved as CMSketch	4	The CM sketch table depth, which equals the number of hash functions.
Width of CM sketch table	Integer	[1;31]	Required if count table saved as CMSketch	20	The CM sketch table width, which is the number of bits of the range of hash function.

Outputs

NAME	TYPE	DESCRIPTION
Counting transform	ITransform interface	The counting transform.

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.
Error 0005	Exception occurs if parameter is less than a specific value.
Error 0007	Exception occurs if parameter is greater than a specific value.
Error 0009	Exception occurs if Azure storage account name or container name specified incorrectly.
Error 0065	Exception occurs if Azure blob name is specified incorrectly.

EXCEPTION	DESCRIPTION
Error 0011	Exception occurs if passed column set argument does not apply to any of dataset columns.
Error 0049	Exception occurs in the case when it is not possible to parse a file.
Error 1000	Internal library exception.
Error 0059	Exception occurs if a column index specified in a column picker cannot be parsed.
Error 0060	Exception occurs when an out of range column range is specified in a column picker.
Error 0089	Exception occurs when the specified number of classes is less than the actual number of classes in a dataset used for counting.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Learning with Counts](#)

Export Count Table

3/10/2021 • 2 minutes to read • [Edit Online](#)

Exports the count table from a saved transformation for use with new data

Category: [Learning with Counts](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Export Count Table** module in Azure Machine Learning Studio (classic). The **Export Count Table** module is provided for backward compatibility with experiments that use the deprecated Build Count Table and deprecated Count Featurizer modules.

When you use the new [Build Counting Transform](#) module to create count-based features, the module outputs both a featurized dataset and a *transform* that creates features from counts. By using the **Export Count Table** module, you can separate the count-based features output by this newer module into **count metadata** and a **count table**. These output formats were used by earlier, now deprecated modules:

For general information about count tables and how they are used to create features, see [Learning with Counts](#).

For all new experiments, we recommend that you use the following modules:

- [Build Counting Transform](#)
- [Modify Count Table Parameters](#)
- [Merge Count Transform](#)

How to configure Export Count Table

1. In Azure Machine Learning Studio (classic), open the experiment where you want to use the imported count table.
2. Locate the saved count transformation, and add it to the experiment.
3. Connect the output of the saved count transformation (labeled **transformation**) to **Export Count Table**.
4. Add the Count Featurizer (deprecated) module to the experiment, and connect it to the two outputs of **Export Count Table**.
5. The Count Featurizer (deprecated) module requires an additional input, for the dataset you want to featurize. Connect the dataset to apply the saved transformation to outputs.
6. Set any necessary parameters for Count Featurizer (deprecated), including the label column, the count columns, the columns to featurize, and the features to output.

You must select a subset of the columns that were originally selected for the counting transformation.

However, the **Export Count Table** module does not provide the list of these columns, so you should review the original experiment and make a note of which columns were used. If you select a column that was not used when creating the transformation, an error is raised.

Examples

Explore examples of count-based featurization using these sample experiments in the [Azure AI Gallery](#):

- [Flight delay prediction](#): Shows how count-based featurization can be useful in a very large dataset.
- [Learning with Counts: Multiclass classification with NYC taxi data](#): Demonstrates the use of count-based features in a multiclass prediction task.
- [Learning with Counts: Binary classification with NYC taxi data](#): Uses count-based features in a binary classification task.

NOTE

If you open a Gallery experiment created using the deprecated versions of the [Learning with Counts](#) modules, the experiment is automatically upgraded to use the newer modules.

Expected inputs

NAME	TYPE	DESCRIPTION
Counting transform	ITransform interface	The counting transform.

Outputs

NAME	TYPE	DESCRIPTION
Dracula count metadata	Data Table	The metadata of the counts.
Dracula count table	Data Table	The count table.

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0086	Exception occurs when a counting transform is invalid.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Learning with Counts](#)

Import Count Table

3/10/2021 • 4 minutes to read • [Edit Online](#)

Imports a previously created table of counts

Category: [Learning with Counts](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Import Count Table** module in Azure Machine Learning Studio (classic).

The purpose of the **Import Count Table** module is to allow customers who created a table of count-based statistics using an earlier version of Azure Machine Learning to upgrade their experiment. This module merges the existing count tables with new data.

For general information about count tables and how they are used to create features, see [Learning with Counts](#).

IMPORTANT

This module is provided solely for backward compatibility with experiments that use the deprecated Build Count Table and deprecated Count Featurizer modules. We recommend that you upgrade your experiment to use the newer modules, to take advantage of new features.

For all new experiments, we recommend that you use the following modules:

- [Build Counting Transform](#)
- [Modify Count Table Parameters](#)
- [Merge Count Transform](#)

How to configure Import Count Table

1. In Azure Machine Learning Studio (classic), open an experiment that contains a count table created using the deprecated Build Count Table module.
2. Add the **Import Count Table** module to the experiment.
3. Connect the two outputs of the Build Count Table (deprecated) module to the matching input ports of the **Import Count Table**.

If you have another dataset of counts that you want to merge with the imported count table, connect it to the rightmost input for the **Import Count Table** module.

4. Use the **Counting type** option to specify where and how the count table is stored:
 - **Dataset:** The data used to build counts is saved as a dataset in Azure Machine Learning Studio (classic).

- **Blob:** The data used to build counts is stored as a block blob in Windows Azure storage.
- **MapReduce:** The data used to build counts is stored as a blob in Windows Azure storage.

This option is typically preferred for very large datasets. To access the counts, you must activate the HDInsight cluster. A MapReduce job is launched to perform the counting. Both of these activities can incur storage and compute costs.

For more information, see [HDInsight on Azure](#).

After specifying the data storage mode, you may need to provide additional connection information for the data, even if you previously used a [Import Data](#) module in the experiment to access data. That is because the Count Featurizer (deprecated) module accesses the data storage separately in order to read the data and build the required tables.

5. Use the **Count table type** option to specify the format and storage mode of the table used to store counts.

- **Dictionary:** Uses a dictionary count table.

All column values in the selected columns are treated as strings, and are hashed using a bit array of up to 31 bits in size. Therefore, all column values are represented by a non-negative 32-bit integer.

- **CMSketch:** Uses a table saved in the *count minimum sketch table*.

With this format, multiple independent hash functions with a smaller range are used to improve memory efficiency and reduce the chance of hash collisions.

In general, you should use the **Dictionary** option for smaller data sets (<1GB), and use the **CMSketch** option for larger datasets.

6. Run the experiment.
7. When complete, right-click the output of the **Import Count Table** module, select **Save as Transform**, and type a name for the transformation. When you do this, the merged count tables and any featurization parameters you might have applied are saved in a format that can be applied to a new dataset.

Examples

Explore examples of count-based featurization using these sample experiments in the [Azure AI Gallery](#):

- [Flight delay prediction](#): Shows how count-based featurization can be useful in a very large dataset.
- [Learning with Counts: Multiclass classification with NYC taxi data](#): demonstrates the use of count-based features in a multiclass prediction task.
- [Learning with Counts: Binary classification with NYC taxi data](#): Uses count-based features in a binary classification task.

NOTE

These Gallery experiments were all created using the earlier, and now deprecated, version of the [Learning with Counts](#) modules. When you open the experiment in Studio (classic), the experiment is automatically upgraded to use the newer modules.

Expected inputs

NAME	TYPE	DESCRIPTION
Count metadata	Data Table	The metadata of the counts
Count table	Data Table	The count table
Counted data set	Data Table	The data set used for counting

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Counting type	CountingType		Required		The counting type

Outputs

NAME	TYPE	DESCRIPTION
Counting transform	ITransform interface	The counting transform

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0018	Exception occurs if input dataset is not valid.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Learning with Counts](#)

Merge Count Transform

3/10/2021 • 3 minutes to read • [Edit Online](#)

Creates a set of features based on a counts table

Category: [Learning with Counts](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Merge Count Transform** module in Azure Machine Learning Studio (classic), to combine two sets of count-based features. By merging two sets of related counts and features, you can potentially improve the coverage and distribution of the features.

Learning from counts is particularly useful in large data sets with high-cardinality features. The ability to combine multiple datasets into count-based feature sets without having to reprocess the datasets makes it easier to gather statistics on very large datasets and apply them to new datasets. For example, count tables can be used to collect information over terabytes of data. You can re-use those statistics to improve the accuracy of predictive models on small data sets.

To merge two sets of count-based features, the features must have been created using tables that have the same schema: that is, both sets must use the same columns, and have the same names and data types.

How to configure Merge Count Transform

1. To use **Merge Count Transform**, you must have created at least one count-based transformation, and that transformation must be present in your workspace. If you saved a count-based transformation from a different experiment, look in the **Transforms** group. If you created the transformation in the current experiment, connect the outputs of the following modules:

- [Build Counting Transform](#). Creates a new count-based transformation from source data.
- [Modify Count Table Parameters](#). Takes an existing count transformation as an input and outputs an updated transformation.
- [Import Count Table](#). This module supports backward compatibility with older experiments that used count-based learning. If you used [Import Count Table](#) to analyze the distribution of values in a dataset, and then converted the values to features using the deprecated Count Featurizer module, use [Import Count Table](#) to convert the results to a transformation.

2. Add the **Merge Count Transform** module to the experiment, and connect a transformation to each input.

TIP

The second transformation is an optional input – you can connect the same transformation twice, or connect nothing on the second input port.

3. If you do not want the second dataset to be weighted equally with the first, specify a value for **Decay factor**. The value that you type indicates how the set of features from the second transformation should be weighted.

For example, the default value of 1 weights both sets of features equally. A value of .5 means that the features in the second set would have half the weight of those in the first set.

4. Optionally, add an instance of the [Apply Transformation](#) module, and apply the transformation to a dataset.

Examples

For examples of how this module is used, see the [Azure AI Gallery](#):

- [Learning with Counts: Binary Classification](#): Demonstrates how to use the learning with counts modules to generate features from columns of categorical values for a binary classification model.
- [Learning with Counts: Multiclass classification with NYC taxi data](#): Demonstrates how to use the learning with counts modules for performing multiclass classification on the publicly available NYC taxi dataset. The sample uses a multiclass logistic regression learner to model this problem.
- [Learning with Counts: Binary classification with NYC taxi data](#): Demonstrates how to use the learning with counts modules for performing binary classification on the publicly available NYC taxi dataset. The sample uses a two-class logistic regression learner to model the problem.

Expected inputs

NAME	TYPE	DESCRIPTION
Previous counting transform	ITransform interface	The counting transform to edit
New counting transform	ITransform interface	The counting transform to add (optional)

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DESCRIPTION	DEFAULT
Decay factor	Float		Required	1.0f	The decay factor to be multiplied to the counting transform at the right input port

Outputs

NAME	TYPE	DESCRIPTION
Merged counting transform	ITransform interface	The merged transform

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0086	Exception occurs when a counting transform is invalid.

See also

[Learning with Counts](#)

Modify Count Table Parameters

3/10/2021 • 4 minutes to read • [Edit Online](#)

Modifies the parameters used to create features from counts

Category: [Learning with Counts](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Modify Count Table Parameters** module in Azure Machine Learning Studio (classic), to change the way that features are generated from a count table.

In general, to create count-based features, you use [Build Counting Transform](#) to process a dataset and create a count table, and from that count table generate a new set of features.

However, if you have already created a count table, you can use the **Modify Count Table Parameters** module to edit the definition of how the count data is processed. This lets you create a different set of count-based statistics based on the existing data, without having to re-analyze the dataset.

How to configure Modify Count Parameters

1. Locate the transformation you want to modify, in the **Transforms** group, and add it to your experiment.

You should have previously run an experiment that created a count transformation.

- **To modify a saved transform:** Locate the transformation, in the **Transforms** group, and add it to your experiment.
- **To modify a count transformation created within the same experiment:** If the transformation has not been saved, but is available as an output in the current experiment (for example, check the output of the [Build Counting Transform](#) module), you can use it directly by connecting the modules.

2. Add the **Modify Count Table Parameters** module and connect the transformation as an input.

3. In the **Properties** pane of the **Modify Count Table Parameters** module, type a value to use as **theGarbage bin threshold**.

This value specifies the minimum number of occurrences that must be found for each feature value, in order for counts to be used. If the frequency of the value is less than the garbage bin threshold, the value-label pair is not counted as a discrete item; instead, all items with counts lower than the threshold value are placed in a single "garbage bin".

If you are using a small dataset and you are counting and training on the same data, a good starting value is 1.

4. For **Additional prior pseudo examples**, type a number that indicates the number of additional pseudo

examples to include. You do not need to provide these examples; the pseudo examples are generated based on the prior distribution.

5. For **Laplacian noise scale**, type a positive floating-point value that represents the scale used for introducing noise sampled from a Laplacian distribution. When you set a scale value, some acceptable level of noise is incorporated into the model, so the model is less likely to be affected by unseen values in data.
6. In **Output features include**, choose the method to use when creating count-based features for inclusion in the transformation.
 - **CountsOnly**: Create features using counts.
 - **LogOddsOnly**: Create features using the log of the odds ratio.
 - **BothCountsAndLogOdds**: Create features using both counts and log odds.
7. Select the **Ignore back off column** option if you want to override the `IsBackOff` flag in the output when creating features. When you select this option, count-based features are created even if the column doesn't have significant count values.
8. Run the experiment. You can then save the output of **Modify Count Table Parameters** as a new transformation, if desired.

Examples

For examples of how this module, see the [Azure AI Gallery](#):

- [Learning with Counts: Binary Classification](#): Demonstrates how to use the learning with counts modules to generate features from columns of categorical values for a binary classification model.
- [Learning with Counts: Multiclass classification with NYC taxi data](#):sample Demonstrates how to use the learning with counts modules for performing multiclass classification on the publicly available NYC taxi dataset. The sample uses a multiclass logistic regression learner to model this problem.
- [Learning with Counts: Binary classification with NYC taxi data](#): Demonstrates how to use the learning with counts modules for performing binary classification on the publicly available NYC taxi dataset. The sample uses a two-class logistic regression learner to model this problem.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

It is statistically safe to count and train on the same data set if you set the Laplacian noise scale parameter.

Expected inputs

NAME	TYPE	DESCRIPTION
Counting transform	ITransform interface	The counting transform to apply

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Garbage bin threshold	Float	$>=0.0f$	Required	10.0f	The threshold under which a column value will be featurized against the garbage bin
Additional prior pseudo examples	Float	$>=0.0f$	Required	42.0f	The additional pseudo examples following prior distributions to be included
Laplacian noise scale	Float	$>=0.0f$	Required	0.0f	The scale of the Laplacian distribution from which noise is sampled
Output features include	OutputFeatureType		Required	BothCountsAndLogOdds	The features to output
Ignore back off column	Boolean		Required	false	Whether to ignore the IsBackOff column in the output

Outputs

NAME	TYPE	DESCRIPTION
Modified transform	ITransform interface	The modified transform

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0086	Exception occurs when a counting transform is invalid.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Learning with Counts](#)

Data Transformation - Manipulation

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that you can use for basic data manipulation.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Machine Learning Studio (classic) supports tasks that are specific to machine learning, such as normalization or feature selection. The modules in this category are intended for more general tasks.

Data manipulation tasks

The modules in this category are intended to support core data management tasks that might need to be performed in Machine Learning Studio (classic). The following tasks are examples of core data management tasks:

- Combine two datasets, either by using joins, or by merging columns or rows.
- Create new categories to use in grouping data.
- Modify column headings, change column data types, or flag columns as features or labels.
- Check for missing values, and then replace them with appropriate values.

Related tasks

- Perform sampling or divide a dataset into training and testing sets: Use the [Data Transformation - Sample and Split](#) modules.
- Scale numbers, normalize data, or put numerical values into bins: Use the [Data Transformation - Scale and Reduce](#) modules.
- Perform calculations on numeric data fields or to generate commonly used statistics: Use the tools in [Statistical Functions](#).

Examples

For examples of how to work with complex data in machine learning experiments, see these samples in the [Azure AI Gallery](#):

- [Data Processing and Analysis](#): Demonstrates key tools and processes.
- [Breast cancer detection](#): Illustrates how to partition datasets, and then apply special processing to each partition.

Modules in this category

The **Data Transformation - Manipulation** category includes the following modules:

- [Add Columns](#): Adds a set of columns from one dataset to another.

- [Add Rows](#): Appends a set of rows from an input dataset to the end of another dataset.
- [Apply SQL Transformation](#): Runs a SQLite query on input datasets to transform the data.
- [Clean Missing Data](#): Specifies how to handle values that are missing from a dataset. This module replaces Missing Values Scrubber, which has been deprecated.
- [Convert to Indicator Values](#): Converts categorical values in columns to indicator values.
- [Edit Metadata](#): Edits metadata that's associated with columns in a dataset.
- [Group Categorical Values](#): Groups data from multiple categories into a new category.
- [Join Data](#): Joins two datasets.
- [Remove Duplicate Rows](#): Removes duplicate rows from a dataset.
- [Select Columns in Dataset](#): Selects columns to include in a dataset or exclude from a dataset in an operation.
- [Select Columns Transform](#): Creates a transformation that selects the same subset of columns as in a specified dataset.
- [SMOTE](#): Increases the number of low-incidence examples in a dataset by using synthetic minority oversampling.

See also

- [Data Transformation](#)
- [Module categories and descriptions](#)
- [A-Z module list](#)

Add Columns

3/10/2021 • 2 minutes to read • [Edit Online](#)

Adds a set of columns from one dataset to another

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Add Columns](#) module in Azure Machine Learning Studio (classic) to concatenate two datasets.

You combine all columns from the two datasets that you specify as inputs to create a single dataset. If you need to concatenate more than two datasets, use several instances of **Add Columns**.

When combining two datasets that contain a different number of rows, we recommend using the [Join Data](#) module, which supports outer joins on a common key column.

How to configure Add Columns

1. Add the **Add Columns** module to your experiment.
2. Connect the two datasets that you want to concatenate. If you want to combine more than two datasets, you can chain together several combinations of **Add Columns**.
 - It is possible to combine two columns that have a different number of rows. The output dataset is padded with missing values for each row in the smaller source column.
 - You cannot choose individual columns to add. All the columns from each dataset are concatenated when you use **Add Columns**. Therefore, if you want to add only a subset of the columns, use [Select Columns in Dataset](#) to create a dataset with the columns you want.
3. Run the experiment.

Results

After the experiment has run:

- To see the first rows of the new dataset, right-click the output of **Add Columns** and select **Visualize**.
- To save and name the concatenated dataset, right-click the output and select **Save as Dataset**.

The number of columns in the new dataset equals the sum of the columns of both input datasets.

If there are two columns with the same name in the input datasets, a numeric suffix is added to the name of the column from the dataset used in the right input column. For example, if there are two instances of a column named **TargetOutcome**, the right column would be renamed **TargetOutcome (1)**.

Examples

For examples of how **Add Columns** is used in an experiment, see the [Azure AI Gallery](#):

- **Customer relationship prediction:** A column that contains labels is combined with a feature dataset.
- **Breast cancer detection:** Datasets that contain features are cleaned and then combined by using [Add Rows](#), [Add Columns](#), and [Join Data](#).

Expected inputs

NAME	TYPE	DESCRIPTION
Left dataset	Data Table	Left dataset
Right dataset	Data Table	Right dataset

Output

NAME	TYPE	DESCRIPTION
Combined dataset	Data Table	Combined dataset

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	An exception occurs if one or more input datasets is null or empty.
Error 0017	An exception occurs if one or more specified columns has a type that is unsupported by the current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Manipulation](#)
[Data Transformation](#)
[A-Z Module List](#)

Add Rows

3/10/2021 • 2 minutes to read • [Edit Online](#)

Appends a set of rows from an input dataset to the end of another dataset

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Add Rows** module in Azure Machine Learning to concatenate two datasets. In concatenation, the rows of the second dataset are added to the end of the first dataset.

Concatenation of rows is useful in scenarios such as these:

- You have generated a series of evaluation statistics, and you want to combine them into one table for easier reporting.
- You have been working with different datasets, and you want to combine the datasets to create a final dataset.

How to use Add Rows

To concatenate rows from two datasets, the rows must have exactly the same schema. This means, the same number of columns, and the same type of data in the columns.

1. Drag the **Add Rows** module into your experiment. You can find it under **Data Transformation**, in the **Manipulate** category.
2. Connect the datasets to the two input ports. The dataset that you want to append should be connected to the second (right) port.
3. Run the experiment. The number of rows in the output dataset should equal the sum of the rows of both input datasets.

If you add the same dataset to both inputs of the **Add Rows** module, the dataset is duplicated.

Technical notes

This section describes implementation details and common questions.

- You cannot filter the source dataset when adding rows. All the rows from both datasets provided as inputs are concatenated when you use **Add Rows**.
- If you want to add only a few rows, use [Partition and Sample](#) to define a condition by which to filter the rows and generate a dataset with only the rows you want.

Examples

To see examples of how this module is used, see the [Azure AI Gallery](#):

- **Demand estimation:** Combines the result of evaluating multiple models into a single dataset and passes it to an Execute R Script for custom processing
- **Breast cancer detection:** Datasets that contain useful features are cleaned and then combined by using [Add Rows](#), [Add Columns](#), and [Join Data](#).
- **Prediction of student performance:** Uses [Add Rows](#) to combine the results of custom metrics that are computed by using [Apply Math Operation](#).
- **Time Series Forecasting:** Uses R scripts to generate custom metrics and then combines them in a single table by using [Add Rows](#).

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset1	Data Table	Dataset rows to be added to the output dataset first
Dataset2	Data Table	Dataset rows to be appended to the first dataset

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset that contains all rows of input datasets

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	An exception occurs if one or more of input datasets is null or empty.
Error 0010	An exception occurs if input datasets have column names that should match but do not.
Error 0016	An exception occurs if input datasets passed to the module should have compatible column types but do not.
Error 0008	An exception occurs if the parameter is not in range.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Manipulation](#)

[Data Transformation](#)

[A-Z Module List](#)

Apply SQL Transformation

3/10/2021 • 8 minutes to read • [Edit Online](#)

Runs a **SQLite** query on input datasets to transform the data

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Apply SQL Transformation** module in Azure Machine Learning Studio (classic), to specify a SQL query on an input dataset or datasets.

SQL is handy when you need to modify your data in complex ways, or persist the data for use in other environments. For example, using the **Apply SQL Transformation** module, you can:

- Create tables for results and save the datasets in a portable database.
- Perform custom transformations on data types, or create aggregates.
- Execute SQL query statements to filter or alter data and return the query results as a data table.

IMPORTANT

The SQL engine used in this module is [SQLite](#). If you are unfamiliar with SQLite syntax, be sure to read the [syntax and usage](#) section of this article for examples.

What is SQLite?

SQLite is a public domain relational database management system that is contained in a C programming library. SQLite is a popular choice as an embedded database for local storage in web browsers.

SQLite was originally designed in 2000 for the U.S. Navy, to support serverless transactions. It is a self-contained database engine that has no management system and hence requires no configuration or administration.

How to configure Apply SQL Transformation

The module can take up to three datasets as inputs. When you reference the datasets connected to each input port, you must use the names `t1`, `t2`, and `t3`. The table number indicates the index of the input port.

The remaining parameter is a SQL query, which uses the SQLite syntax. This module supports all standard statements of the SQLite syntax. For a list of unsupported statements, see the [Technical Notes](#) section.

General syntax and usage

- When typing multiple lines in the **SQL Script** text box, use a semi-colon to terminate each statement. Otherwise, line breaks are converted to spaces.

For example, the following statements are equivalent:

```
SELECT  
*  
from  
t1;
```

```
SELECT * from t1;
```

- You can add comments by using either `--` at the beginning of each line, or by enclosing text using `/* */`.

For example, this statement is valid:

```
SELECT * from t1  
/*WHERE ItemID BETWEEN 1 AND 100*/;
```

- If a column name duplicates the name of a reserved keyword, syntax highlighting is applied to the text inside the **SQL Script** text box. To avoid confusion, you should enclose column names with square brackets (to follow the Transact-SQL convention) or backticks or double quotation marks (the ANSI SQL convention).

For example, in the following query on the Blood Donation dataset, **Time** is a valid column name but is also a reserved keyword.

```
SELECT Recency, Frequency, Monetary, Time, Class  
FROM t1  
WHERE Time between 3 and 20;
```

If you run the query as is, the query might return the correct results, but depending on the dataset, it might return an error. Here are some examples of how to avoid the issue:

```
-- Transact-SQL  
SELECT [Recency], [Frequency], [Monetary], [Time], [Class]  
FROM t1  
WHERE [Time] between 3 and 20;  
-- ANSI SQL  
SELECT "Recency", "Frequency", "Monetary", "Time", "Class"  
FROM t1  
WHERE `Time` between 3 and 20;
```

NOTE

Syntax highlighting remains on the keyword even after it is enclosed in quotes or brackets.

- SQLite is **case insensitive**, except for a few commands that have case-sensitive variants with different meanings (GLOB vs. glob).

SELECT statement

In the `SELECT` statement, column names that include spaces or other characters prohibited in identifiers must be enclosed in double quotation marks, square brackets, or backtick characters (`).

For example, this query references the Two-Class Iris dataset on `t1`, but one column name contains a

prohibited character, so the column name is enclosed in quotation marks.

```
SELECT class, "sepal-length" FROM t1;
```

You can add a `WHERE` clause to filter values in the dataset.

```
SELECT class, "sepal-length" FROM t1 WHERE "sepal-length" >5.0;
```

The SQLite syntax does not support the `TOP` keyword, which is used in Transact-SQL. Instead, you can use the `LIMIT` keyword, or a `FETCH` statement.

For example, compare these queries on the Bike Rental dataset.

```
-- unsupported in SQLite
SELECT TOP 100 [dteday] FROM t1 ;
ORDER BY [dteday] DESC;

-- Returns top 100
SELECT [dteday] FROM t1 LIMIT 100 ;
ORDER BY [dteday] DESC;

-- Returns top 100. Note that FETCH is on a new line.
SELECT [dteday] FROM t1 - ;
FETCH FIRST 100 rows ONLY;
ORDER BY [dteday] DESC;
```

Joins

The following examples use the Restaurant Ratings dataset on the input port corresponding to `t1`, and the Restaurant Features dataset on the input port corresponding to `t2`.

The following statement joins the two tables to create a dataset that combines the specified restaurant features with average ratings for each restaurant.

```
SELECT DISTINCT(t2.placeid),
t2.name, t2.city, t2.state, t2.price, t2.alcohol,
AVG(rating) AS 'AvgRating'
FROM t1
JOIN t2
ON t1.placeID = t2.placeID
GROUP BY t2.placeid;
```

Aggregate functions

This section provides basic examples of some common SQL aggregate functions, using SQLite.

Aggregate functions currently supported are: `AVG` , `COUNT` , `MAX` , `MIN` , `SUM` , `TOTAL` .

The following query returns a dataset containing the restaurant ID, along with the average rating for the restaurant.

```
SELECT DISTINCT placeid,
AVG(rating) AS 'AvgRating',
FROM t1
GROUP BY placeid
```

Working with strings

SQLite supports the double pipe operator for concatenating strings.

The following statement creates a new column by concatenating two text columns.

```
SELECT placeID, name,
(city || '-' || state) AS 'Target Region',
FROM t1
```

WARNING

The Transact-SQL string concatenation operator is not supported: `+` ([String Concatenation](#)). For example, the expression `('city + '-' + state) AS 'Target Region'` in the example query would return 0 for all values.

However, even though the operator is not supported for this data type, no error is raised in Azure Machine Learning. Be sure to verify the results of [Apply SQL Transformation](#) before using the resulting dataset in an experiment.

COALESCE and CASE

`COALESCE` evaluates multiple arguments, in order, and returns the value of the first expression that does not evaluate to NULL.

For example, this query on the Steel Annealing Multi-Class dataset returns the first non-null flag from a list of columns assumed to have mutually exclusive values. If no flag is found, the string "none" is returned.

```
SELECT classes, family, [product-type],
COALESCE(bt,bc,bf,[bw/me],bl, "none") AS TemperType
FROM t1;
```

The `CASE` statement is useful for testing values and returning a new value based on the evaluated results. SQLite supports the following syntax for `CASE` statements:

- `CASE WHEN [condition] THEN [expression] ELSE [expression] END`
- `CASE [expression] WHEN [value] THEN [expression] ELSE [expression] END`

For example, suppose you had previously used the [Convert to Indicator Values](#) module to create a set feature columns containing true-false values. The following query collapses the values in multiple feature columns into a single multivalued column.

```
SELECT userID, [smoker-0], [smoker-1],
CASE
WHEN [smoker-0]= '1' THEN 'smoker'
WHEN [smoker-1]= '1' THEN 'nonsmoker'
ELSE 'unknown'
END AS newLabel
FROM t1;
```

Examples

For an example of how this module might be used in machine learning experiments, see this sample in the [Azure AI Gallery](#):

- [Apply SQL Transformation](#): Uses the Restaurant Ratings, Restaurant Features, and Restaurant Customers dataset to illustrate simple joins, select statements, and aggregate functions.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

- An input is always required on port 1.
- If the input dataset has column names, the columns in the output dataset will use the column names from the input dataset.

If the input dataset does not have column names, the column names in the table are automatically created by using the following naming convention: T1COL1, T1COL2, T1COL3, and so on, where the numbers indicate the index of each column in the input dataset.
- For column identifiers that contain a space or other special characters, always enclose the column identifier in square brackets or double quotation marks when referring to the column in the `SELECT` or `WHERE` clauses.

Unsupported statements

Although SQLite supports much of the ANSI SQL standard, it does not include many features supported by commercial relational database systems. For more information, see [SQL as Understood by SQLite](#). Also, be aware of the following restrictions when creating SQL statements:

- SQLite uses dynamic typing for values, rather than assigning a type to a column as in most relational database systems. It is weakly typed, and allows implicit type conversion.
- `LEFT OUTER JOIN` is implemented, but not `RIGHT OUTER JOIN` or `FULL OUTER JOIN`.
- You can use `RENAME TABLE` and `ADD COLUMN` statements with the `ALTER TABLE` command, but other clauses are not supported, including `DROP COLUMN`, `ALTER COLUMN`, and `ADD CONSTRAINT`.
- You can create a `VIEW` within SQLite, but thereafter views are read-only. You cannot execute a `DELETE`, `INSERT`, or `UPDATE` statement on a view. However, you can create a trigger that fires on an attempt to `DELETE`, `INSERT`, or `UPDATE` on a view and perform other operations in the body of the trigger.

In addition to the list of non-supported functions provided on the official SQLite site, the following wiki provides a list of other unsupported features: [SQLite - Unsupported SQL](#)

Expected inputs

NAME	TYPE	DESCRIPTION
Table1	Data Table	Input dataset1
Table2	Data Table	Input dataset2
Table3	Data Table	Input dataset3

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
SQL Query Script	any	StreamReader		SQL query statement

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Output dataset

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	An exception occurs if one or more specified columns of the dataset couldn't be found.
Error 0003	An exception occurs if one or more of the input datasets is null or empty.
Error 0069	SQL logic error or missing database

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

- [Manipulation](#)
- [Data Transformation](#)
- [A-Z Module List](#)

Clean Missing Data

3/10/2021 • 16 minutes to read • [Edit Online](#)

Specifies how to handle the values missing from a dataset

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Clean Missing Data](#) module in Azure Machine Learning Studio (classic), to remove, replace, or infer missing values.

Data scientists often check data for missing values and then perform various operations to fix the data or insert new values. The goal of such cleaning operations is to prevent problems caused by missing data that can arise when training a model.

This module supports multiple type of operations for "cleaning" missing values, including:

- Replacing missing values with a placeholder, mean, or other value
- Completely removing rows and columns that have missing values
- Inferring values based on statistical methods

TIP

New to machine learning? This article provides a good explanation of why you would use each of the different methods for replacing missing values: [Methods for handling missing values](#)

Using this module does not change your source dataset. Instead, it creates a new dataset in your workspace that you can use in the subsequent workflow. You can also save the new, cleaned dataset for reuse.

This module also outputs a definition of the transformation used to clean the missing values. You can re-use this transformation on other datasets that have the same schema, by using the [Apply Transformation](#) module.

How to use Clean Missing Data

This module lets you define a cleaning operation. You can also save the cleaning operation so that you can apply it later to new data. See the following links for a description of how to create and save a cleaning process:

- [To replace missing values](#)
- [To apply a cleaning transformation to new data](#)

IMPORTANT

The cleaning method that you use for handling missing values can dramatically affect your results. We recommend that you experiment with different methods. Consider both the justification for use of a particular method, and the quality of the results.

Replace missing values

Each time that you apply the [Clean Missing Data](#) module to a set of data, the same cleaning operation is applied to all columns that you select. Therefore, if you need to clean different columns using different methods, use separate instances of the module.

1. Add the [Clean Missing Data](#) module to your experiment, and connect the dataset that has missing values.
2. For **Columns to be cleaned**, choose the columns that contain the missing values you want to change. You can choose multiple columns, but you must use the same replacement method in all selected columns. Therefore, typically you need to clean string columns and numeric columns separately.

For example, to check for missing values in all numeric columns:

- a. Open the Column Selector, and select **WITH RULES**.
- b. For **BEGIN WITH**, select **NO COLUMNS**.

You can also start with **ALL COLUMNS** and then exclude columns. Initially, rules are not shown if you first click **ALL COLUMNS**, but you can click **NO COLUMNS** and then click **ALL COLUMNS** again to start with all columns and then filter out (exclude) columns based on the name, data type, or columns index.

- c. For **Include**, select **Column type** from the dropdown list, and then select **Numeric**, or a more specific numeric type.

Any cleaning or replacement method that you choose must be applicable to **all** columns in the selection. If the data in any column is incompatible with the specified operation, the module returns an error and stops the experiment.

3. For **Minimum missing value ratio**, specify the minimum number of missing values required for the operation to be performed.

You use this option in combination with **Maximum missing value ratio** to define the conditions under which a cleaning operation is performed on the dataset. If there are too many or too few rows that are missing values, the operation cannot be performed.

The number you enter represents the **ratio** of missing values to all values in the column. By default, the **Minimum missing value ratio** property is set to 0. This means that missing values are cleaned even if there is only one missing value. For an example of how to use this option, see [Setting a Threshold for Cleaning Operations](#).

WARNING

This condition must be met by each and every column in order for the specified operation to apply. For example, assume you selected three columns and then set the minimum ratio of missing values to .2 (20%), but only one column actually has 20% missing values. In this case, the cleanup operation would apply only to the column with over 20% missing values. Therefore, the other columns would be unchanged.

If you have any doubt about whether missing values were changed, select the option, **Generate missing value indicator column**. A column is appended to the dataset to indicate whether or not each column met the specified criteria for the minimum and maximum ranges.

4. For **Maximum missing value ratio**, specify the maximum number of missing values that can be present for the operation to be performed.

For example, you might want to perform missing value substitution only if 30% or fewer of the rows contain missing values, but leave the values as-is if more than 30% of rows have missing values.

You define the number as the ratio of missing values to all values in the column. By default, the **Maximum missing value ratio** is set to 1. This means that missing values are cleaned even if 100% of the values in the column are missing.

NOTE

When you set a threshold using the options **Minimum missing value ratio** or **Maximum missing value ratio**, the cleaning operation cannot be performed if even one of the selected columns does not meet the criteria.

5. For **Cleaning Mode**, select one of the following options for replacing or removing missing values:

- **Replace using MICE:** For each missing value, this option assigns a new value, which is calculated by using a method described in the statistical literature as "Multivariate Imputation using Chained Equations" or "Multiple Imputation by Chained Equations". With a *multiple imputation method*, each variable with missing data is modeled conditionally using the other variables in the data before filling in the missing values. In contrast, in a *single imputation method* (such as replacing a missing value with a column mean) a single pass is made over the data to determine the fill value.

All imputation methods introduce some error or bias, but multiple imputation better simulates the process generating the data and the probability distribution of the data.

For a general introduction to methods for handling missing values, see [Missing Data: the state of the art. Schafer and Graham, 2002.](#)

WARNING

This option cannot be applied to completely empty columns. Such columns must be removed or passed to the output as is.

- **Custom substitution value:** Use this option to specify a placeholder value (such as a 0 or NA) that applies to all missing values. The value that you specify as a replacement must be compatible with the data type of the column.
- **Replace with mean:** Calculates the column mean and uses the mean as the replacement value for each missing value in the column.

Applies only to columns that have Integer, Double, or Boolean data types. See the [Technical Notes](#) section for more information.

- **Replace with median:** Calculates the column median value, and uses the median value as the replacement for any missing value in the column.

Applies only to columns that have Integer or Double data types. See the [Technical notes](#) section for more information.

- **Replace with mode:** Calculates the mode for the column, and uses the mode as the replacement value for every missing value in the column.

Applies to columns that have Integer, Double, Boolean, or Categorical data types. See the [Technical Notes](#) section for more information.

- **Remove entire row:** Completely removes any row in the dataset that has one or more missing values. This is useful if the missing value can be considered randomly missing.
- **Remove entire column:** Completely removes any column in the dataset that has one or more missing values.
- **Replace using Probabilistic PCA:** Replaces the missing values by using a linear model that analyzes the correlations between the columns and estimates a low-dimensional approximation of the data, from which the full data is reconstructed. The underlying dimensionality reduction is a probabilistic form of Principal Component Analysis (PCA), and it implements a variant of the model proposed in the Journal of the Royal Statistical Society, Series B 21(3), 611–622 by Tipping and Bishop.

Compared to other options, such as Multiple Imputation using Chained Equations (MICE), this option has the advantage of not requiring the application of predictors for each column. Instead, it approximates the covariance for the full dataset. Therefore, it might offer better performance for datasets that have missing values in many columns.

The key limitations of this method are that it expands categorical columns into numerical indicators and computes a dense covariance matrix of the resulting data. It also is not optimized for sparse representations. For these reasons, datasets with large numbers of columns and/or large categorical domains (tens of thousands) are not supported due to prohibitive space consumption.

TIP

Remember that the method you choose is applied to all columns in the selection. Thus, if you want to replace some missing values with zeroes in some columns but insert a placeholder in other columns, you should use [Select Columns in Dataset](#) to separate the data and use different instances of the [Clean Missing Data](#) module.

6. The option **Replacement value** is available if you have selected the option, **Custom substitution value**. Type a new value to use as the replacement value for all missing values in the column.

Note that you can use this option only in columns that have the Integer, Double, Boolean, or Date data types. For date columns, the replacement value can also be entered as the number of 100-nanosecond ticks since 1/1/0001 12:00 A.M.

7. **Generate missing value indicator column:** Select this option if you want to output some indication of whether the values in the column met the criteria for missing value cleaning. This option is particularly useful when you are setting up a new cleaning operation and want to make sure it works as designed.
8. Run the experiment, or select the [Clean Missing Data](#) module and click **Run selected**.

Results

The module returns two outputs:

- **Cleaned dataset:** A dataset comprised of the selected columns, with missing values handled as specified, along with an indicator column, if you selected that option.
Columns not selected for cleaning are also "passed through".
- **Cleaning transformation:** A data transformation used for cleaning, that can be saved in your workspace and applied to new data later.

Apply a saved cleaning operation to new data

If you need to repeat cleaning operations often, we recommend that you save your recipe for data cleansing as a

transform, to reuse with the same dataset. Saving a cleaning transformation is particularly useful if you must frequently re-import and then clean data that has the same schema.

1. Add the [Apply Transformation](#) module to your experiment.
2. Add the dataset you want to clean, and connect the dataset to the right-hand input port.
3. Expand the **Transforms** group in the left-hand pane of Studio (classic). Locate the saved transformation and drag it into the experiment.
4. Connect the saved transformation to the left input port of [Apply Transformation](#).

When you apply a saved transformation, you cannot select the columns to which the transformation are applied. That is because the transformation has been already defined and applies automatically to the data types specified in the original operation.

However, suppose you created a transformation on a subset of numeric columns. You can apply this transformation to a dataset of mixed column types without raising an error, because the missing values are changed only in the matching numeric columns.

5. Run the experiment.

Examples

See examples of how this module is used in the [Azure AI Gallery](#):

- [Prediction of student performance](#): In this sample, zeros are inserted for missing values.
- [Cross Validation for Binary Classifier sample](#): Zeros are used to fill-in for missing values, and an indicator column is created to track the changes. Columns with all missing values are also retained.
- [Dataset Processing and Analysis](#): In this sample, different branches of the experiment use different methods for missing value substitution, and the datasets are then evaluated by using [Summarize Data](#) and [Compute Linear Correlation](#).
- [Flight delay prediction sample](#): Empty rows are removed entirely.

Technical notes

This section contains implementation details, as well as known issues and commonly asked questions.

- An error occurs if the mean or median option is used when any string columns are selected. If you need to process columns of different data types, create two instances of [Clean Missing Data](#).
- When replacing missing values with a mean value in columns with the Boolean, Integer, DateTime, or TimeSpan data types, the column is first converted to floating point numbers, the mean is calculated, and then the result is rounded to the nearest value of the original data type.
- When you type a replacement value, the value must be compatible with the data type in the selected column.
- Values of `NaN`, `Inf`, and `-Inf` are allowed for columns where the data type is Double.
- When using the MICE method, the replacement value is predicted by using the trained MICE model.
- Using [Clean Missing Data](#) can reset other column types to **feature**. If your data contains other types of columns, such as labels, use [Edit Metadata](#) to correct the column types.

Restrictions on use of cleaning transformations

The following restrictions apply when you use a saved transformation (based on [Clean Missing Data](#)) to new

data:

- A saved transformation cannot generate indicator values, even if this option was used in the original cleaning operation. Consider the indicator values as most useful when testing a new transformation.
- The transformation does not calculate new values based on the new dataset. In other words, if you used [Clean Missing Data](#) on Dataset A and generated a mean value of 0.5, that same value would be applied as the mean for replacing missing values in Dataset B, regardless of the actual values in Dataset B.
- The data type of the columns in the new dataset must match the data type of the columns on which the transformation was originally created. An error is raised if any operations are performed on the column that implicitly change the data type.

For example, suppose you create a mean for an integer data column [Col1], and save the transformation. Now you want to apply the cleanup transformation to a copy of [Col1] that has been adjusted using a formula, such as $([Col1] / 1.5)$. To ensure that the result is an integer, you round up the result, but still get an error when you apply the transformation. However, if you adjust the value using a formula such as $([Col 1] * 10)$, no error is raised!

To avoid such issues, use [Edit Metadata](#) to explicitly reset the data type to integer. In general, operations in [Apply Math Operation](#) module implicitly change numeric columns to `double`.

Setting and interpreting threshold values

When you specify a threshold for cleaning operations using the options **Minimum missing value ratio** or **Maximum missing value ratio**, the results can be unexpected or confusing. To illustrate how the options for maximum and minimum missing values work, we have provided some examples from the [Automobile Prices](#) sample dataset, which has many columns with missing values.

The following table shows the count of missing values for several columns in that dataset, together with the ratio of missing values computed on the dataset. The ratio of missing values (in the rightmost column) is the value that would be used in evaluating the dataset against the specified threshold values.

Assume that you set **Minimum missing value ratio** to 0.019 and set **Maximum missing value ratio** to 0.020. Given the following table of values, some columns meet the threshold criteria, and some do not:

- The columns `bore` and `stroke` meet the threshold criteria.
- The columns `normalized-losses` and `compression-ratio` do not meet the threshold criteria.

COLUMN NAME	COUNT OF MISSING VALUES	RATIO OF MISSING VALUES
Normalized-losses	41	0.2
Bore	4	0.019512195
Stroke	4	0.019512195
Compression ratio	0	0

Because **some** columns in the selection did not meet the specified criteria, no cleaning operation was performed on any column. To help you figure out what happened, the module returns the value **FALSE** in the two indicator columns, `bore_IsMissing` and `stroke_IsMissing`.

However, if you change the threshold back to the default values of 0 for **Minimum missing value ratio** and 1 for **Maximum missing value ratio**, an indicator column is returned for all selected columns, and the specified operation is performed.

TIP

If you are uncertain about whether missing value clean-up is working as expected, select the [Generate missing value indicator column](#) option.

Known issues

If you use the MICE method to clean data and then process a dataset that contains missing values, you might get the following error: "AFx Library library exception: Model is not trained. (Error 1000)"

This error occurs only when the MICE method is selected, and if the training dataset does not contain missing values but the test dataset does.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Dataset to be cleaned

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Columns to be cleaned	Any	ColumnSelection	All	Select columns for the missing values clean operation.
Minimum missing value ratio	[0.0;1.0]	Float	0.0	Clean only column with missing value ratio above the specified value, out of a set of all selected columns.
Maximum missing value ratio	[0.0;1.0]	Float	1.0	Clean only columns with missing value ratio below the specified value out of a set of all selected columns.
Cleaning mode	List	Handling policy	Custom substitution value	Choose an algorithm to use when cleaning missing values.
Replacement value	Any	String	"0"	Type a value to take the place of missing values. This value is optional.
Cols with all missing values	Any	ColumnsWithAllValuesMissing	Remove	Indicate if columns of all missing values should be preserved in the output.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Generate missing value indicator column	Any	Boolean	false	Generate a column that indicates which rows were cleaned.
Number of iterations	[1;10]	Integer	5	Specify the number of iterations when using MICE.
Number of iterations for PCA prediction	[1;50]	Integer	10	Specify the number of iterations when using a PCA prediction.

Outputs

NAME	TYPE	DESCRIPTION
Cleaned dataset	Data Table	Cleaned dataset
Cleaning transformation	ITransform interface	Transformation that is to be passed to the Apply Transformation module to clean new data.

Exceptions

EXCEPTION	DESCRIPTION
Error 0002	An exception occurs if one or more parameters could not be parsed or converted from the specified type into the type required by the target method.
Error 0003	An exception occurs if one or more input datasets are null or empty.
Error 0008	An exception occurs if a parameter is not in range.
Error 0013	An exception occurs if the leaner passed to the module has an invalid type.
Error 0018	An exception occurs if the input dataset is not valid.
Error 0039	An exception occurs if the operation fails.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

- [Manipulation](#)
- [Data Transformation](#)
- [A-Z Module List](#)

Convert to Indicator Values

3/10/2021 • 4 minutes to read • [Edit Online](#)

Converts categorical values in columns to indicator values

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Convert to Indicator Values](#) module in Azure Machine Learning Studio (classic). The purpose of this module is to convert columns that contain categorical values into a series of binary indicator columns that can more easily be used as features in a machine learning model.

How to configure Convert to Indicator Values

1. Add the [Convert to Indicator Values](#) module to your Azure Machine Learning experiment, and connect it to the dataset containing the columns you want to convert. You can find this module under **Data Transformations**, in the **Manipulation** category.
2. Use the **Column Selector** to choose one or more categorical columns.

To ensure that the columns you select are categorical, use [Edit Metadata](#) before [Convert to Indicator Values](#) in your experiment, to mark the target column as categorical.

3. Select the **Overwrite categorical columns** option if you want to output only the new Boolean columns.

By default, this option is off, which lets you see the categorical column that is the source, together with the related indicator columns.

TIP

If you choose the option to overwrite, the source column is not actually deleted or modified. Instead, the new columns are generated and presented in the output dataset, and the source column remains available in the workspace. If you need to see the original data, you can use the [Add Columns](#) module at any time to add the source column back in.

4. Run the experiment.

Results

For example, suppose you have a column with scores that indicate whether a server has a high, medium or low probability of failure.

SERVER ID	FAILURE SCORE
10301	Low
10302	Medium
10303	High

When you apply [Convert to Indicator Values](#), the single column of labels is converted into multiple columns containing Boolean values:

SERVER ID	FAILURE SCORE - LOW	FAILURE SCORE - MEDIUM	FAILURE SCORE - HIGH
10301	1	0	0
10302	0	1	0
10303	0	0	1

Here is how the conversion works:

- In the **Failure score** column that describes risk, there are only three possible values (High, Medium, and Low), and no missing values. Therefore exactly three new columns are created.
- The new indicator columns are named based on the column headings and values of the source column, using this pattern: *<source column>- <data value>*.
- There should be a 1 in exactly one indicator column, and 0 in all other indicator columns. That is because each server can have only one risk rating.

You can now use the three indicator columns as features and analyze their correlation with other properties that are associated with different risk level.

Examples

To see examples of how this module is used, see the [Azure AI Gallery](#):

- **Breast cancer detection:** Patients are binned into groups based on patient ID numbers, and then **Indicator Values** is used to flag which group the patient belongs to. Later, the group indicators are used when scoring models.
- **Direct marketing:** Probabilities are compared to a constant by using [Apply Math Operation](#), and the Yes/No values that indicate whether the score was above or below the constant are turned into new indicator columns.
- **Network intrusion detection:** Log data is loaded from Azure storage. The class variable (which describes, for example, if an attack is a rootkit or buffer overflow) is converted to a categorical column and then expanded to multiple indicator values.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Usage tips

- Only columns that are marked as categorical can be converted to indicator columns. If you see this error, it is likely that one of the columns you selected is not categorical:

Error 0056: Column with name <column name> is not in an allowed category.

By default most string columns are handled as string features, so you must explicitly mark them as categorical using [Edit Metadata](#).

- An error is displayed if you do not select at least one categorical column.
- There is no limit on the number of columns that you can convert to indicator columns. However, because each column of values can yield multiple indicator columns, you might want to convert and review just a few columns at a time.
- If the column contains missing values, a separate indicator column is created for the missing category, with this name: <source column>- Missing
- If the column that you convert to indicator values contains numbers, they must be marked as categorical like any other feature column. After you have done so, the numbers are treated as discrete values. For example, if you have a numeric column with MPG values ranging from 25 to 30, a new indicator column would be created for each discrete value:

MAKE	HIGHWAY MPG -25	HIGHWAY MPG -26	HIGHWAY MPG -27	HIGHWAY MPG -28	HIGHWAY MPG -29	HIGHWAY MPG -30
Alfa Romeo	0	0	0	0	0	1

To avoid getting a huge number of indicator columns, we recommend that you first check the number of values in the column, and bin or quantize the data appropriately.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Dataset with categorical columns

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Categorical columns to convert	Any	ColumnSelection		Select categorical columns to convert to indicator matrices.
Overwrite categorical columns	Any	Boolean	false	If True, overwrite the selected categorical columns; otherwise, append the resulting indicator matrices to the dataset.

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with categorical columns converted to indicator matrices.

See also

[Manipulation](#)
[Data Transformation](#)
[A-Z Module List](#)

Edit Metadata

3/10/2021 • 11 minutes to read • [Edit Online](#)

Edits metadata associated with columns in a dataset

Category: Data Transformation / Manipulation

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Edit Metadata](#) module in Azure Machine Learning Studio (classic) to change metadata that is associated with columns in a dataset. The values and the data types in the dataset are not actually altered; what changes is the metadata inside Azure Machine Learning that tells downstream components how to use the column.

Typical metadata changes might include:

- Treating Boolean or numeric columns as categorical values
- Indicating which column contains the *class* label, or the values you want to categorize or predict
- Marking columns as features
- Changing date/time values to a numeric value, or vice versa
- Renaming columns

Use [Edit Metadata](#) any time you need to modify the definition of a column, typically to meet requirements for a downstream module. For example, some modules can work only with specific data types, or require flags on the columns, such as `IsFeature` or `IsCategorical`.

After performing the required operation, you can reset the metadata to its original state.

How to configure Edit Metadata

1. In Azure Machine Learning Studio (classic), add [Edit Metadata](#) module to your experiment and connect the dataset you want to update. You can find it under **Data Transformation**, in the **Manipulate** category.
2. Click **Launch the column selector** and choose the column or set of columns to work with. You can choose columns individually, by name or index, or you can choose a group of columns, by type.

TIP

Need help using column indices? See the [Technical Notes](#) section.

3. Select the **Data type** option if you need to assign a different data type to the selected columns. Changing

the data type might be needed for certain operations: for example, if your source dataset has numbers handled as text, you must change them to a numeric data type before using math operations.

- The data types supported are `String`, `Integer`, `Floating point`, `Boolean`, `DateTime`, and `TimeSpan`.
- If multiple columns are selected, you must apply the metadata changes to **all** selected columns. For example, let's say you choose 2-3 numeric columns. You could change them all to a string data type, and rename them in one operation. However, you can't change one column to a string data type and another column from a float to an integer.
- If you do not specify a new data type, the column metadata is unchanged.
- Changes of data type affect only the metadata that is associated with the dataset and how the data is handled in downstream operations. The actual column values are not altered unless you perform a different operation (such as rounding) on the column. You can recover the original data type at any time by using [Edit Metadata](#) to reset the column data type.

NOTE

If you change any type of number to the `DateTime` type, leave the `DateTime Format` field blank. Currently, it is not possible to specify the target data format.

Azure Machine Learning can convert dates to numbers, or numbers to dates, if the numbers are compatible with one of the supported .NET DateTime objects. For more information, see the [Technical Notes](#) section.

4. Select the **Categorical** option to specify that the values in the selected columns should be treated as categories.

For example, you might have a column that contains the numbers 0,1 and 2, but know that the numbers actually mean "Smoker", "Non smoker" and "Unknown". In that case, by flagging the column as categorical you can ensure that the values are not used in numeric calculations, only to group data.

5. Use the **Fields** option if you want to change the way that Azure Machine Learning uses the data in a model.

- **Feature:** Use this option to flag a column as a feature, for use with modules that operate only on feature columns. By default, all columns are initially treated as features.
- **Label:** Use this option to mark the label (also known as the predictable attribute, or target variable). Many modules require that at least one (and only one) label column be present in the dataset.

In many cases, Azure Machine Learning can infer that a column contains a class label, but by setting this metadata you can ensure that the column is identified correctly. Setting this option does not change data values, only the way that some machine learning algorithms handle the data.

- **Weight:** Use this option with numeric data to indicate that column values represent weights for use in machine learning scoring or training operations. Only one weight column can be present in a dataset, and the column must be numeric.

TIP

Have data that doesn't fit into these categories? For example, your dataset might contain values such as unique identifiers that are not useful as variables. Sometimes IDs can cause problems when used in a model.

Fortunately "under the covers" Azure Machine Learning keeps all your data, so you don't have to delete such columns from the dataset. When you need to perform operations on some special set of columns, just remove all other columns temporarily by using the [Select Columns in Dataset](#) module. Later you can merge the columns back into the dataset by using the [Add Columns](#) module.

6. Use the following options to clear previous selections and restore metadata to the default values.

- **Clear feature:** Use this option to remove the feature flag.

Because all columns are initially treated as features, for modules that perform mathematical operations, you might need to use this option to prevent numeric columns from being treated as variables.

- **Clear label:** Use this option to remove the **label** metadata from the specified column.
- **Clear score:** Use this option to remove the **score** metadata from the specified column.

Currently the ability to explicitly mark a column as a score is not available in Azure Machine Learning. However, some operations result in a column being flagged as a score internally. Also, a custom R module might output score values.

- **Clear weight:** Use this option to remove the **weight** metadata from the specified column.

7. For **New column names**, type the new name of the selected column or columns.

- Column names can use only characters that are supported by the UTF-8 encoding. Empty strings, nulls, or names consisting entirely of spaces are not allowed.
- To rename multiple columns, type the names as a comma-separated list in order of the column indices.
- All selected columns must be renamed. You cannot omit or skip columns.

TIP

If you need to rename multiple columns, you can paste in a comma-delimited string prepared in advance. Or, use the [Execute R Script](#) or [Apply SQL Transformation](#) modules. See the [Technical Notes](#) section for code and examples.

8. Run the experiment.

Examples

For examples of how [Edit Metadata](#) is used in preparing data and building models, see the [Azure AI Gallery](#):

- **Breast cancer detection:** Column names are changed after joining to datasets. The *Patient ID* column is also flagged as **categorical** to ensure that it is not used in a calculation, but rather than handled as a string value.
- **Twitter sentiment analysis:** Demonstrates how to use [Edit Metadata](#) to ensure that columns are treated as features. Later in the experiment, the feature metadata is cleared.
- **Data Processing and analysis:** In this sample, [Edit Metadata](#) is used to define new column names for data that was loaded from a webpage.

Technical notes

This section contains known issues, frequently asked questions, and some examples of common workarounds.

Known Issues

- **Custom metadata is not supported.** It is not possible to use custom metadata in Azure Machine Learning or to edit column metadata outside [Edit Metadata](#). For example, you cannot add metadata indicating that a column is a unique identifier, or add other descriptive attributes . Azure Machine Learning supports only the metadata attributes that are used within R for working with factors, features, weights, and labels.
- **Unsupported data types.** The following numeric data types are not supported: Double (decimal) and TimeStamp.
- **Identifying score columns.** Currently there is no option in [Edit Metadata](#) to flag a column as containing *scores*. However, you can use the [Execute R Script](#) module with a script similar to the following to indicate that a column contains scores:

```
dataset <- maml.mapInputPort(1)
attr(dataset$x, "label.type")= "True Labels"
attr(dataset$y, "feature.channel")= "Multiclass Classification Scores"
attr(dataset$y, "score.type")= "Assigned Labels"
maml.mapOutputPort("dataset");
```

- **Problems with datetime formats.** The underlying `datetime` data type used by Azure Machine Learning is `POSIXct`.

If all dates in a column can be parsed by the default parser, the column is imported and treated as string data.

If you try to convert a column to `DateTime` by using the [Edit Metadata](#) module and get an error, it means that the date is not in a format that .Net accepts by default. In this case, we recommend that you use the [Execute R Script](#) module or the [Apply SQL Transformation](#) module to transform your column to a format that is accepted by the default parser.

[DateTime.Parse Method](#)

[Standard Date and Time Format Strings](#)

Selecting columns using column indices

In very large datasets, it is not feasible to manually type or select all column names. Using the column index is one shortcut that you can use to specify many columns. This section provides some tips on using column indices.

For example, open the Column Selector, click **WITH RULES**, select **Include** and **column indices**, and then type a range or series of numbers as follows:

- Type `1-20` to select the first 20 columns
- Type `5-20` to select a range of columns beginning at 5 and including column 20.
- Type `1,5,10,15` to select discontinuous columns
- Type `1-2, 5` to select columns 1, 2 and 5, skipping columns 3 and 4
- You cannot type an index value that is greater than the number of columns available in the dataset.

The following experiments provide some examples of other methods for selecting and modifying multiple columns:

- [Binary Classification: Breast Cancer Detection](#): The original data contained many blank columns generated during import from a spreadsheet. The extra columns were removed by specifying columns 1-11 in the **Split Data** module.
- [Download dataset from UCI](#): Demonstrates how you can provide column names as a list using the **Enter Data Manually** module, and then insert the list into the dataset as headings, using the **Execute R script** module.
- [Regex Select Columns](#): This experiment provides a custom module that lets you apply a regular expression to column names. You could use this module as an input to **Edit Metadata**.

Alternate methods for modifying column names

If you have many columns to rename, you can use the **Execute R Script** module, or the **Apply SQL Transformation** module.

Using R script

Data sets used by Azure Machine Learning are passed into this module as a `data.frame`, meaning that you can use the R `colnames()` function and other related R functions, to list or change column names.

For example, the following code creates a list of new column names, and then applies that list to the input dataset to generate new column headings.

```
irisdata <- maml.mapInputPort(1);
newnames <- c("CLASS", "SEPAL LENGTH", "SEPAL WIDTH", "PETAL LENGTH", "PETAL WIDTH");
colnames(irisdata) = newnames
maml.mapOutputPort("irisdata");
```

The following example uses a regular expression in R to globally replace all instances of the specified string in the column names for `irisdata`:

```
# Map input dataset to variable
newirisdata <- maml.mapInputPort(1) # class: data.frame
names(newirisdata) <- gsub("col", "iris", names(newirisdata))
maml.mapOutputPort("newirisdata");
```

Using SQL

The following example takes a dataset as the input, and then changes the column names using the **AS** keyword.

```
SELECT col1 as [C1],
       col2 as [C2],
       col3 as [C3],
       col4 as [C4],
       col5 as [C5]
  FROM t1;
```

Expected input

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Column	Any	ColumnSelection		Choose the columns to which your changes should apply.
Data type	List	Metadata editor datatype	Unchanged	Specify the new data type for the column.
Categorical	List	Metadata editor categorical	Unchanged	Indicate if the column should be flagged as categorical.
Fields	List	Metadata editor flag	Unchanged	Specify if the column should be considered a feature or label by learning algorithms.
<i>New column names</i>	any	String		Type the new names for the columns.

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with changed metadata

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	An exception occurs if one or more of input datasets are null or empty.
Error 0017	An exception occurs if one or more specified columns have a type that is unsupported by the current module.
Error 0020	An exception occurs if the number of columns in some of the datasets that are passed to the module is too small.
Error 0031	An exception occurs if the number of columns in the column set is less than needed.
Error 0027	An exception occurs when two objects have to be of the same size, but they are not.
Error 0028	An exception occurs when the column set contains duplicate column names and it is not allowed.
Error 0037	An exception occurs if multiple label columns are specified and only one is allowed.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Manipulation](#)

[Data Transformation](#)

[A-Z Module List](#)

Group Categorical Values

3/10/2021 • 5 minutes to read • [Edit Online](#)

Groups data from multiple categories into a new category

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Group Categorical Values** module in Azure Machine Learning Studio (classic), to create an in-place lookup table.

The typical use for grouping categorical values is to merge multiple string values into a single new level. For example, you might assign individual postal codes in a region to a single regional code, or group multiple products under one category.

To use this module, you type the lookup values you want to use, and map existing values to the replacement values. You can create groupings only for categorical columns, not to columns of numeric type or columns designated as labels or features.

Any column values that are not explicitly mapped to a new level are assigned to a default level. For example, if you did not map all the individual postal codes, they would be grouped in a level for unmapped values, which you might name **Unknown**.

NOTE

A maximum of 20 new levels can be created, including the default level. If you need more values, or need to define mappings dynamically, we recommend that you use custom R script in the [Execute R Script](#) module. Or, use SQL statements in the [Apply SQL Transformation](#) module.

How to use Group Categorical Values

We recommend that you prepare the list of existing values, and the new categories, beforehand. For each category, you should prepare a new category name, and a comma-separated list of values to include in the category.

1. Add the **Group Categorical Values** module to your experiment. You can find the module under **Data Transformation, Manipulation**.
2. Connect a dataset that has the values you want to transform.
3. In the **Properties** pane of **Group Categorical Values**, use the Column Selector to choose the column that has the levels you want to reduce.
 - We recommend that you click **BEGIN WITH** and **NO COLUMNS** to start, and then add columns

by name. Otherwise too many columns might be added as candidates, leading to an error.

- The column must be a categorical column. If it is not, add [Edit Metadata](#) upstream, and change the column type.
 - Be sure to remove from the input any columns to which string replacement should not be applied.
4. For **Output mode**, indicate whether you want to output just the new levels, or append the changes to see the original column, with the replacements side by side.
- The default, **ResultOnly**, shows only the new values. The **Inplace** option replaces the existing column values with the new levels.
5. For **Default level name**, type a string value to use as the replacement for all values that are not explicitly mapped. You might use something such as "Unknown" or "Default".

NOTE

This default level value is applied to all values that cannot be mapped. If you accidentally included columns that you did not intend to map, the value would be applied to all values in the columns. Therefore, check that column selection is accurate before processing.

6. For **New number of levels**, type a number that indicates the total number of new categories (levels), including the default level for unmapped values.
7. For **Name of new level 1**, provide the new group name for the first category.
8. In the text box that immediately follows, **Comma-separated list of old levels to map to new level 1**, type or paste an exhaustive list of all values to map to the new level. Wildcard characters and regular expressions are not allowed.
9. Continue to type new level names and type or paste values that should be mapped to the new level.

We recommend that you save your list of values in a separate file as you are working. If you change the number of levels, any strings that you previously typed are removed, and you must start over.

However, if you are editing a module that was previously saved, you can revert to the original settings.

10. Run the experiment.

Results

To view the results, right-click the **Group Categorical Values** module, select **Results dataset**, and click **Visualize**.

Examples

For examples of machine learning in action, see the [Azure AI Gallery](#).

You can also try this module for yourself, by using a small dataset with some string variables that can be easily grouped, such as the **Automobile price** dataset that is provided in Azure Machine Learning Studio (classic).

Let's assume that you want to group cars in the Automobile price dataset by engine size, using the number of cylinders. Rather than lots of different engine sizes, you will create the new levels, "big", "small", and "other" as follows:

- Big engines: six cylinders or larger
- Small engines: two or four cylinders
- Other: anything else

1. Add the **Select Columns in Dataset** module, and select only the `num-of-cylinders` column.
2. Add the **Edit Metadata** module, and change the `num-of-cylinders` column to **Categorical**.
3. Add the **Group Categorical Values** module and connect the modified dataset.
4. For **Default level name**, type `other`. You don't need to provide values for this level.
5. For **Name of new level 1**, type `big`. In the list of old levels to map to level 1, paste in `six, eight, twelve`.
6. For **Name of new level 2**, type `small`. For the mapped values, paste in `two, four`.
7. Run the experiment.
8. When you **Visualize** the results, you realize that the original dataset had some odd engine sizes that you didn't account for, such as `five` and `three`. All such items are mapped to the `other` level.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

- You might encounter the error message, "Column with name "<columnname>" is not in an allowed category."

This message indicates that the column you selected is not a categorical column. You can mark the column as `categorical` by using [Edit Metadata](#), or select a different column that contains appropriate category values.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Data to group

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Selected columns	any	ColumnSelection	CategoricalAll	Select the columns that will be grouped.
Output mode	any	OutputTo	ResultOnly	Specify how the category labels should be output.
Default level name	any	String		Indicate the default level to use if no mappings match.
New number of levels	List	Number of groups		Specify the number of levels after values have been grouped, including the default level.

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Grouped data

See also

[Manipulation](#)
[Data Transformation](#)
[A-Z Module List](#)

Join Data

3/10/2021 • 6 minutes to read • [Edit Online](#)

Joins two datasets

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Join Data](#) module in Azure Machine Learning Studio (classic) to merge two datasets using a database-style *join operation*.

To perform a join on two datasets, they must be related by a **single** key column. Composite keys are not supported.

How to configure Join Data

1. In Azure Machine Learning Studio (classic), add the datasets you want to combine, and then drag the **Join Data** module into your experiment.

You can find the module in the **Data Transformation** category, under **Manipulation**.

2. Connect the datasets to the **Join Data** module.

The **Join Data** module does not support a right outer join, so if you want to ensure that rows from a particular dataset are included in the output, that dataset must be on the lefthand input.

3. Click **Launch column selector** to choose a single key column for the dataset on the left input.

4. Click **Launch column selector** to choose a single key column for the dataset on the right input.

5. Select the **Match case** option if you are joining on a text column and want to ensure that the join preserves case sensitivity.

For example, if you select this option, `A1000` would be considered a different key value than `a1000`.

If you deselect this option, case sensitivity is **not** enforced, and `A1000` would be considered the same as `a1000`.

6. Use the **Join type** dropdown list to specify how the datasets should be combined. types:

- **Inner Join:** An *inner join* is the typical join operation. It returns the combined rows only when the values of the key columns match.
- **Left Outer Join:** A *left outer join* returns joined rows for all rows from the left table. When a row in the left table has no matching rows in the right table, the returned row contains missing values for all columns that come from the right table unless you specify a replacement value for missing values.

- **Full Outer Join:** A *full outer join* returns all rows from the left table (**table1**) and from the right table (**table2**).

For each of the rows in the left table that have no matching rows in the right table, the join results include a row containing missing values from the right table.

For each of the rows in the right table that have no matching rows in the left table, the join results include a row containing missing values for all columns from the left table.

- **Left Semi-Join:** A *left semi-join* returns only the values from the left table when the values of the key columns match.

7. For the option, **Keep right key columns in joined table:**

- Deselect the option to get a single key column in the results.
- Leave the option selected to view the keys from both input tables.

8. Run the experiment, or select the **Join Data** module and selected **Run Selected**, to perform the join.

9. To view the results, right-click the **Join Data** module, select **Results dataset**, and click **Visualize**.

Examples

You can see examples of how this module is used in the [Azure AI Gallery](#):

- [Breast cancer detection](#): **Join Data** is used to combine the positive training cases with the negative training cases after the proportion of cases has been adjusted.
- [Flight delay prediction](#): In this sample, **Join Data** is used to bring together useful features from external datasets.
- [Movie recommendation](#): Two datasets are joined so that we can present the recommended movie titles rather than a movie ID.
- [Prediction of student performance](#): In this sample, **Join Data** is used to bring in new features.

Technical notes

This section describes implementation details, and answers to some frequently asked questions.

Restrictions

- The combined dataset cannot have two columns with the same name. If the left and right datasets have any duplicate column names, a numeric suffix is appended to the column names of the right dataset to make them unique.

For example, if both datasets had a column named Month, the column from the left dataset would remain as is, and the column from the right dataset would be renamed Month (1).

- The algorithm that is used for comparison of key values is hash-forced.
- Each column of the joined dataset preserves a categorical type, if the corresponding column of the input dataset is categorical.
- In left outer joins, if there are any missing values, a categorical level is created in the left dataset for missing values. This is true even if there are no missing values in the joined (right) dataset.

How can I join a table on a composite key?

If you need to join a table that uses composite keys (that is, the primary key relies on two independent columns), use a module such as the following to concatenate the contents of the two key columns:

- [Execute R Script](#)

For example, use code like the following inside the R script to concatenate the first and second columns of the input dataframe using a hyphen as separator. `paste(inputdf$Col1,inputdf$Col2,sep="-")`

- [Apply SQL Transformation](#)

The concatenation operator in SQLite is `||`.

How can I join tables that don't have a key?

If your dataset has no key column, you can still combine it with another dataset, either by generating a key, or by using the [Add Columns](#) module.

The [Add Columns](#) module behaves like R, and can merge two datasets on a row-by-row basis, if the datasets have the same number of rows. An error is raised if the datasets are of a different size.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset1	Data Table	First dataset to join
Dataset2	Data Table	Second dataset to join

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Join key columns for L	Any	ColumnSelection		Select the join key columns for the first dataset.
Join key columns for R	Any	ColumnSelection		Select the join key columns for the second dataset.
Match case	Any	Boolean	True	Indicate whether a case-sensitive comparison is allowed in key columns.
Join type	List	Type	Inner join	Choose a join type.
Keep right key columns in joined table	Any	Boolean	True	Indicate whether to keep key columns from the second dataset in the joined dataset.

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Result of join operation

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	An exception occurs if one or more specified columns of the dataset couldn't be found.
Error 0003	An exception occurs if one or more inputs are null or empty.
Error 0006	An exception occurs if the parameter is greater than or equal to the specified value.
Error 0016	An exception occurs if the input datasets that are passed to the module should have compatible column types, but they do not.
Error 0017	An exception occurs if one or more specified columns have types that are unsupported by the current module.
Error 0020	An exception occurs if the number of columns in some of the datasets that are passed to the module is too small.
Error 0028	An exception occurs when the column set contains duplicate column names and it is not allowed.
Error 0011	An exception occurs if the argument for the passed column set does not apply to any dataset columns.
Error 0027	An exception occurs when two objects have to be of the same size, but they are not.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

- [Manipulation](#)
- [Data Transformation](#)
- [A-Z Module List](#)

Remove Duplicate Rows

3/10/2021 • 5 minutes to read • [Edit Online](#)

Removes the duplicate rows from a dataset

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Remove Duplicate Rows** module in Azure Machine Learning Studio (classic), to remove potential duplicates from a dataset.

For example, assume your data looks like the following, and represents multiple records for patients.

PATIENTID	INITIALS	GENDER	AGE	ADMITTED
1	F.M.	M	53	Jan
2	F.A.M.	M	53	Jan
3	F.A.M.	M	24	Jan
3	F.M.	M	24	Feb
4	F.M.	M	23	Feb
	F.M.	M	23	
5	F.A.M.	M	53	

Clearly, this example has multiple columns with potentially duplicate data. Whether they are actually duplicates depends on your knowledge of the data.

- For example, you might know that many patients have the same name. You wouldn't eliminate duplicates using any name columns, only the **ID** column. That way, only the rows with duplicate ID values are filtered out, regardless of whether the patients have the same name or not.
- Alternatively, you might decide to allow duplicates in the ID field, and use some other combination of fields to find unique records, such as first name, last name, age, and gender.

To set the criteria for whether a row is duplicate or not, you specify a single column or a set of columns to use as **keys**. Two rows are considered duplicates only when the values in **all** key columns are equal.

When you run the module, it creates a candidate dataset, and returns a set of rows that have no duplicates across the set of columns you specified.

IMPORTANT

The source dataset is not altered; this module creates a new dataset that is filtered to exclude duplicates, based on the criteria you specify.

How to use Remove Duplicate Rows

1. Add the module to your experiment. You can find the **Remove Duplicate Rows** module under **Data Transformation, Manipulation**.
2. Connect the dataset that you want to check for duplicate rows.
3. In the **Properties** pane, under **Key column selection filter expression**, click **Launch column selector**, to choose columns to use in identifying duplicates.

In this context, **Key** does not mean a unique identifier. All columns that you select using the Column Selector are designated as **key columns**. All un-selected columns are considered non-key columns. The combination of columns that you select as keys determines the uniqueness of the records. (Think of it as a SQL statement that uses multiple equality joins.)

Examples:

- "I want to ensure that IDs are unique": Choose only the ID column.
- "I want to ensure that the combination of first name, last name, and ID is unique": Select all three columns.

4. Use the **Retain first duplicate row** checkbox to indicate which row to return when duplicates are found:
 - If selected, the first row is returned and others discarded.
 - If you uncheck this option, the last duplicate row is kept in the results, and others are discarded.

See the [Technical notes](#) section for information on how missing values are handled.

5. Run the experiment, or click the module and select **Run Selected**.
6. To review the results, right-click the module, select **Results dataset**, and click **Visualize**.

TIP

If the results are difficult to understand, or if you want to exclude some columns from consideration, you can remove columns by using the [Select Columns in Dataset](#) module.

Examples

To see examples of how this module is used, see the [Azure AI Gallery](#):

- [Breast cancer detection](#): **Remove Duplicate Rows** is used to consolidate the training and test datasets after adding feature columns.
- [Movie recommendation](#): Uses **Remove Duplicate Rows** to ensure that there is only one user *rating* per movie.
- [Twitter sentiment analysis](#): **Remove Duplicate Rows** is applied to only the ID and popularity columns, to ensure that there is only one *ordinal ranking* value per movie. In other words, a movie cannot be both 1st and 3rd, so a single value is used even if users ranked the movie differently.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

The module works by looping through all rows of the input dataset. It collects into a candidate output dataset all rows where the unique combination of key column values appears for the first time.

The column array type is preserved independently of the results of row filtering. You cannot force the array to a particular data type by filtering out invalid values; the column array type is based on all values in the column. This restriction also applies when filtering missing values.

The algorithm used for comparing data values is hash-forced.

Missing values

The input dataset might have missing values in non-key columns and key columns. These rules apply to missing values:

- A missing value is considered a valid value in key columns. Missing values can be present in both key.
- In a sparse dataset, the missing value is considered equal only if it equals the default representation of a sparse value.
- In key columns, a missing value is considered equal to other missing values, but not equal to non-missing values.

Expected input

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Key column selection filter expression	any	ColumnSelection		Choose the key columns to use when searching for duplicates.
Retain first duplicate row	any	Boolean	true	Indicate whether to keep the first row of a set of duplicates and discard others. If False, the last duplicate row encountered is kept.

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Filtered dataset

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	An exception occurs if one or more of the input datasets are null or empty.
Error 0020	An exception occurs if the number of columns in some of the datasets passed to the module is too small.
Error 0017	An exception occurs if one or more specified columns have a type that is unsupported by the current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Manipulation](#)

[A-Z Module List](#)

Select Columns in Dataset

3/10/2021 • 9 minutes to read • [Edit Online](#)

Selects columns to include or exclude from a dataset in an operation

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Select Columns in Dataset](#) module in Azure Machine Learning Studio (classic), to choose a subset of columns to use in downstream operations. The module does not physically remove the columns from the source dataset; instead, it creates a subset of columns, much like a database *view* or *projection*.

This module is particularly useful when you need to limit the columns available for a downstream operation, or if you want to reduce the size of the dataset by removing unneeded columns.

The columns in the dataset are output in the same order as in the original data, even if you specify them in a different order.

How to use Select Columns in Dataset

This module has no parameters. You use the column selector to choose the columns to include or exclude.

Choose columns by name

There are multiple options in the module for choosing columns by name:

- Filter and search

Click the **BY NAME** option.

If you have connected a dataset that is already populated, a list of available columns should appear. If no columns appear, you might need to run upstream modules to view the column list.

To filter the list, type in the search box. For example, if you type the letter **w** in the search box, the list is filtered to show the column names that contain the letter **w**.

Select columns and click the right arrow button to move the selected columns to the list in the right-hand pane.

- To select a continuous range of column names, press **Shift + Click**.
- To add individual columns to the selection, press **Ctrl + Click**.

Click the checkmark button to save and close.

- Use names in combination with other rules

Click the **WITH RULES** option.

Choose a rule, such as showing columns of a specific data type.

Then, click individual columns of that type by name, to add them to the selection list.

- Type or paste a comma-separated list of column names

If your dataset is very wide, it might be easier to use indexes or generated lists of names, rather than selecting columns individually. Assuming you have prepared the list in advance:

1. Click the **WITH RULES** option.
2. Select **No columns**, select **Include**, and then click inside the text box with the red exclamation mark.
3. Paste in or type a comma-separated list of previously validated column names. You cannot save the module if any column has an invalid name, so be sure to check the names beforehand.

You can also use this method to specify a list of columns using their index values. See the [Examples](#) section for tips on how to work with column indices.

Choose by type

If you use the **WITH RULES** option, you can apply multiple conditions on the column selections. For example, you might need to get only feature columns of a numeric data type.

The **BEGIN WITH** option determines your starting point and is very important for understanding the results.

- If you select the **ALL COLUMNS** option, all columns are added to the list. Then, you must use the **Exclude** option to *remove* columns that meet certain conditions.

For example, you might start with all columns and then remove columns by name, or by type.

- If you select the **NO COLUMNS** option, the list of columns starts out empty. You then specify conditions to *add* columns to the list.

If you apply multiple rules, each condition is **additive**. For example, say you start with no columns, and then add a rule to get all numeric columns. In the Automobile price dataset, that results in 16 columns. Then, you click the **+** sign to add a new condition, and select **Include all features**. The resulting dataset includes all the numeric columns, plus all the feature columns, including some string feature columns.

Choose by column index

The column index refers to the order of the column within the original dataset.

- Columns are numbered sequentially starting at 1.
- To get a range of columns, use a hyphen.
- Open-ended specifications such as `1-` or `-3` are not allowed.
- Duplicate index values (or column names) are not allowed, and might result in an error.

For example, assuming your dataset has at least eight columns, you could paste in any of the following examples to return multiple non-contiguous columns:

- `8,1-4,6`
- `1,3-8`
- `1,3-6,4`

the final example does not result in an error; however, it returns a single instance of column `4`.

For additional tips on working with column indices, see the [Examples](#) section.

Change order of columns

The option **Allow duplicates and preserve column order in selection** starts with an empty list, and adds columns that you specify by name or by index. Unlike other options, which always return columns in their

"natural order", this option outputs the columns in the order that you name or list them.

For example, in a dataset with the columns Col1, Col2, Col3, and Col4, you could reverse the order of the columns and leave out column 2, by specifying either of the following lists:

- `Col4, Col3, Col1`
- `4,3,1`

Examples

For examples of how to use [Select Columns in Dataset](#), see these sample experiments in the [Model Gallery](#):

- The [Breast cancer detection](#) sample uses [Select Columns in Dataset](#) to remove a trailing empty column, remove a column with duplicate data, and to project training and test sets.
- In the [Flight delay prediction](#) sample, [Select Columns in Dataset](#) is used to exclude all string columns and to exclude columns by name.
- In the [Prediction of student performance](#) sample, [Select Columns in Dataset](#) is used to get all temporal features, and to exclude multiple columns.
- In the [Compare Regressors](#) sample, [Select Columns in Dataset](#) is used to exclude the column, `num-of-doors`, because it is the wrong data type for the math operation that follows.

Common scenarios for column selection

The following examples describe some typical ways that users apply [Select Columns in Dataset](#) in machine learning, and provides some tips for how to select the columns:

- **I want to remove text columns from the dataset so I can apply a math operation to all numeric columns.**

Many operations require that only numeric columns be present in the dataset. You can temporarily remove columns that would cause an error, by excluding text and excluding categorical columns (numbers that represent discrete categories).

1. Click **Launch column selector**.
2. For **Begin With**, select **All columns**.
3. Select the **Exclude** option, select **column type**, and then select **String**.
4. Click the plus sign (+) to add a new condition.
5. Select the **Exclude** option, select **column type**, and then select **Categorical**.

- **I need to apply feature selection to only the categorical feature columns.**

If you need to separate columns of a similar type, you can apply multiple conditions. For example, features can be either categorical or numeric, but some feature selection modules do not allow non-numeric fields, so you first have to get features, and then add a condition to get just the numeric features.

1. Click **Launch column selector**.
2. For **Begin With**, select **No columns**.
3. Select the **Include** option, and select **all features**.
4. Click the plus sign (+) to add a new condition.
5. Select the **Include** option, select **column type**, and then select **Categorical**.

- I need to apply a different normalization operation to different numeric columns.

Before applying mathematical operations, you might need to separate integers from floating point numbers, and so forth. To do this use the data types and apply multiple conditions.

1. Click **Launch column selector**.
2. For **Begin With**, select **No columns**.
3. Select the **Include** option, select **column type**, and then select **Numeric**.
4. Click the plus sign (+) to add a new condition.
5. Select the **Include** option, select **column type**, and then select the numeric type that is incompatible with the downstream operation.

- There are too many columns to choose using the selector.

Often, after importing a dataset, you find that it has a lot of columns that aren't needed for modeling. However, you want to preserve them for output later, or for identifying cases. You can do this by splitting the dataset into two parts (metadata, and columns used for modeling) and later recombine columns as needed, by using [Add Columns](#).

1. Click **Launch column selector**.
2. For **Begin With**, select **No columns**.
3. Select the **Include** option, select **column type**, and then select **Feature**.
4. Click the plus sign (+) to add a new condition.
5. Select the **Include** option, select **column type**, and then select **Label**.
6. Repeat these steps, but start with all columns, and then exclude feature and label columns to create a dataset of only the metadata.

- I don't know the index values for the columns I need.

If there are just a few columns in your dataset, you can use the **Visualize** option to see the first 100 rows and then figure out which column is index 1, 2, and so forth.

- The indexes in Azure Machine Learning start at 1, so the first column is always 1.
- To get the index of the last column, look at the two lists of columns in the Column Selector: AVAILABLE COLUMNS and SELECTED COLUMNS. The gray bar beneath the column list displays the count of columns in each list. Thus, if 24 columns are available and two columns are selected, there are a total of 26 columns, and the index of the final column is 26.

Another option for extracting the schema of your dataset is to use the [Execute R Script](#) module to get the column names with index numbers.

1. Connect your dataset to the **Execute R Script** module.
2. In the module, type a script like the following to output the column names. The line starting with `myindex` generates a sequence that represents the indexes in order.

```
dataset1 <- maml.mapInputPort(1) # class: data.frame
mycolnames <- names(dataset1);
myindex <- seq(from = 1, to = length(mycolnames), by=1);
outdata <- as.data.frame(cbind(myindex, mycolnames));
maml.mapOutputPort("outdata");
```

Results on Automobile price dataset

MYINDEX	MYCOLNAMES
1	symboling
2	normalized-losses
3	make

Technical notes

If you are familiar with relational databases, this module creates a *projection* of the data; hence the original name, **Project Columns**. In database terms, a projection is a function, such as a Transact-SQL or LINQ statement, that takes a data in tabular format as input and produces a related output.

In relational algebra, a projection is a unary operation, which is written as a set of attribute names. The result of a projection is the set of those attributes, with other attributes discarded.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Select columns	any	ColumnSelection		Select columns to keep in the projected dataset.

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Output dataset

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	An exception occurs if one or more specified columns of the dataset couldn't be found.
Error 0003	An exception occurs if one or more input datasets are null or empty.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Manipulation](#)

Select Columns Transform

3/10/2021 • 2 minutes to read • [Edit Online](#)

Creates a transformation that selects the same subset of columns as in the given dataset

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

This article describes how to use the **Select Columns Transform** module in Azure Machine Learning Studio (classic). The purpose of the **Select Columns Transform** module is to ensure that a predictable, consistent set of columns is always used in downstream machine learning operations.

This module is particularly helpful for tasks such as scoring, which require specific columns. Changes in the available columns might break the experiment or change the results.

You use the **Select Columns Transform** to create and save a set of columns. Then, use the [Apply Transformation](#) module to apply those selections to new data.

How to use Select Columns Transform

This scenario assumes that you intend to use feature selection to generate a dynamic set of columns that will be used for training a model. To ensure that column selections are the same for the scoring process, you use the **Select Columns Transform** module to capture the column selections and apply them elsewhere in the experiment.

1. Add an input dataset to your experiment in Studio (classic).
2. Add an instance of [Filter Based Feature Selection](#).
3. Connect the modules and configure the feature selection module to automatically find some number of best features in the input dataset.
4. Add an instance of [Train Model](#) and use the output of [Filter Based Feature Selection](#) as the input for training.

IMPORTANT

Because feature importance is decided based on the values in the column, you cannot know in advance which columns might be available for input to [Train Model](#).

5. Now, attach an instance of the **Select Columns Transform** module.

This generates a column selection as a transformation that can be saved or applied to other datasets. This step ensures that the columns identified by feature selection are saved for reuse by other modules.

6. Add the [Score Model](#) module.

Do not connect the input dataset.

Instead, add the [Apply Transformation](#) module, and connect the output of the feature selection transformation.

IMPORTANT

You cannot expect to apply [Filter Based Feature Selection](#) to the scoring dataset, and get the same results. Since feature selection is based on values, it might choose a different set of columns, which would cause the scoring operation to fail.

7. Run the experiment.

This process of saving and then applying a column selection ensures that the same data schema is available for training and scoring.

Examples

For examples of how to use this module, see the [Azure AI Gallery](#):

- [Select columns transform](#): A complete walkthrough that uses this module.
- [Filter features and remove them from scoring inputs](#): Save this experiment to your workspace to see how the module is used in a complete experimental workflow.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset with desired columns	Data Table	Dataset containing desired set of columns

Outputs

NAME	TYPE	DESCRIPTION
Columns selection transformation	ITransform interface	Transformation that selects the same subset of columns as in the given dataset.

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.

See also

[Manipulation](#)

[Select Columns in Dataset](#)

SMOTE

3/10/2021 • 6 minutes to read • [Edit Online](#)

Increases the number of low incidence examples in a dataset using synthetic minority oversampling

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **SMOTE** module in Azure Machine Learning Studio (classic) to increase the number of underrepresented cases in a dataset used for machine learning. SMOTE is a better way of increasing the number of rare cases than simply duplicating existing cases.

You connect the SMOTE module to a dataset that is *imbalanced*. There are many reasons why a dataset might be imbalanced: the category you are targeting might be very rare in the population, or the data might simply be difficult to collect. Typically, you use SMOTE when the *class* you want to analyze is under-represented.

The module returns a dataset that contains the original samples, plus an additional number of synthetic minority samples, depending on the percentage you specify.

More about SMOTE

SMOTE stands for *Synthetic Minority Oversampling Technique*. This is a statistical technique for increasing the number of cases in your dataset in a balanced way. The module works by generating new instances from existing minority cases that you supply as input. This implementation of SMOTE does **not** change the number of majority cases.

The new instances are not just copies of existing minority cases; instead, the algorithm takes samples of the *feature space* for each target class and its nearest neighbors, and generates new examples that combine features of the target case with features of its neighbors. This approach increases the features available to each class and makes the samples more general.

SMOTE takes the entire dataset as an input, but it increases the percentage of only the minority cases. For example, suppose you have an imbalanced dataset where just 1% of the cases have the target value A (the minority class), and 99% of the cases have the value B. To increase the percentage of minority cases to twice the previous percentage, you would enter 200 for **SMOTE percentage** in the module's properties.

Examples

We recommend that you try using **SMOTE** with a small dataset to see how it works. The following example uses the Blood Donation dataset available in Azure Machine Learning Studio (classic).

If you add the dataset to an experiment, and click **Visualize** on the dataset's output, you can see that, of the 748 rows or cases in the dataset, there are 570 cases (76%) of Class 0, and 178 cases (24%) of class 1. Although this isn't terribly imbalanced, Class 1 represents the people who donated blood, and thus these rows contain the *feature space* that you want to model.

To increase the number of cases, you can set the value of **SMOTE percentage**, using multiples of 100, as follows:

	CLASS 0	CLASS 1	TOTAL
Original dataset	570	178	748
(equivalent to SMOTE percentage = 0)	76%	24%	
SMOTE percentage = 100	570 62%	356 38%	926
SMOTE percentage = 200	570 52%	534 48%	1104
SMOTE percentage = 300	570 44%	712 56%	1282

WARNING

Increasing the number of cases using SMOTE is not guaranteed to produce more accurate models. You should try experimenting with different percentages, different feature sets, and different numbers of nearest neighbors to see how adding cases influences your model.

How to configure SMOTE

1. Add the SMOTE module to your experiment. You can find the module under Data Transformation modules, in the manipulation category.
2. Connect the dataset you want to boost. If you want to specify the feature space for building the new cases, either by using only specific columns, or by excluding some, use the [Select Columns in Dataset](#) module to isolate the columns you want to use before using SMOTE.

Otherwise, creation of new cases using SMOTE is based on **all** the columns that you provide as inputs.
3. Ensure that the column containing the label, or target class, is marked as such.

If there is no label column, use the [Edit Metadata](#) module to select the column that contains the class labels, and select **Label** from the **Fields** dropdown list.
4. The SMOTE module automatically identifies the minority class in the label column, and then gets all examples for the minority class.
5. In the **SMOTE percentage** option, type a whole number that indicates the target percentage of minority cases in the output dataset. For example:
 - You type **0 (%)**. The SMOTE module returns exactly the same dataset that you provided as input, adding no new minority cases. In this dataset, the class proportion has not changed.
 - You type **100 (%)**. The SMOTE module generates new minority cases, adding the same number of minority cases that were in the original dataset. Because SMOTE does not increase the number of majority cases, the proportion of cases of each class has now changed.

- You type 200 (%). The module doubles the percentage of minority cases compared to the original dataset. This **does not** result in having twice as many minority cases as before. Rather, the size of the dataset is increased in such a way that the number of majority cases stays the same, and the number of minority cases is increased till it matches the desired percentage value.

NOTE

Use only multiples of 100 for the SMOTE percentage.

6. Use the **Number of nearest neighbors** option to determine the size of the feature space that the SMOTE algorithm uses when in building new cases. A *nearest neighbor* is a row of data (a case) that is very similar to some target case. The distance between any two cases is measured by combining the weighted vectors of all features.
 - By increasing the number of nearest neighbors, you get features from more cases.
 - By keeping the number of nearest neighbors low, you use features that are more like those in the original sample.
7. Type a value in the **Random seed** textbox if you want to ensure the same results over runs of the same experiment, with the same data. Otherwise the module generates a random seed based on processor clock values when the experiment is deployed, which can cause slightly different results over runs.
8. Run the experiment.

The output of the module is a dataset containing the original rows plus some number of added rows with minority cases.

TIP

If you want to figure out which new rows were added, you can use the [Apply SQL Transformation](#) or [Join Data](#) modules.

Technical notes

- When publishing a model that uses the SMOTE module, remove SMOTE from the predictive experiment before it is published as a web service. The reason is that SMOTE is intended for improving a model during training, and is not intended for scoring. You might get an error if a published predictive experiment contains the SMOTE module.
- You can often get better results if you apply missing value cleaning or other transformations to fix data before applying SMOTE.
- Some researchers have investigated whether SMOTE is effective on high-dimensional or sparse data, such as those used in text classification or genomics datasets. This paper has a good summary of the effects and of the theoretical validity of applying SMOTE in such cases: [Blagus and Lusa: SMOTE for high-dimensional class-imbalanced data](#)

If SMOTE is not effective in your dataset, other approaches that you might consider include various methods for oversampling the minority cases or undersampling the majority cases, as well as ensemble techniques that help the learner directly, by using clustering, bagging, or adaptive boosting.

Expected input

NAME	TYPE	DESCRIPTION
Samples	Data Table	A dataset of samples

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
SMOTE percentage	>=0	Integer	100	Amount of oversampling in multiples of 100.
Number of nearest neighbors	>=1	Integer	1	The number of nearest neighbors from which to draw features for new cases
Random seed	Any	Integer	0	Seed for the random number generator

Output

NAME	TYPE	DESCRIPTION
Table	Data Table	A Data Table containing the original samples plus an additional number of synthetic minority class samples. The number of new samples is $(smotePercent/100)*T$, where T is the number of minority class samples.

See also

[Sample and Split](#)

[A-Z Module List](#)

Data Transformation - Sample and Split

3/10/2021 • 4 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that you can use to partition or sample data.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Splitting and sampling datasets are both important tasks in machine learning. For example, it's a common practice to divide data into training and testing sets to help you evaluate a model on a holdout dataset.

Sampling is also increasingly important in the era of big data, to ensure that there's a fair distribution of classes in your training data. Sampling also helps ensure that you're not processing more data than is needed.

You can use Machine Learning Studio (classic) modules to customize the way you split or sample datasets:

- Filter training data based on an attribute in the data.
- Perform stratified sampling to divide the class variable equally among n number of groups.
- Divide source data into a training and testing data set by using a custom ratio.
- Apply regular expressions to the data to filter out invalid values.

Choosing the right operation: Split or sampling

Machine Learning Studio (classic) provides two modules that encapsulate tasks. The modules sound similar, but they have different uses, and provide complementary functionality. It's likely that you'll use both modules in an experiment, to get the right amount and the right mix of data.

Next, we compare the **Split Data** module and the **Partition and Sample** module by seeing which tasks each module is commonly used for.

Uses of the Split Data module

- **Divide data into two groups.** Use the **Split Data** module. The module produces exactly two splits of the data. You can specify the condition on which the data is split, and the proportion of the data to put into each subset. **Split Data** always saves the subset of data that doesn't meet the conditions.
- **Allocate label values equally to datasets.** The option to stratify on a specified column is supported by both modules. However, if you want to create two datasets and are mostly interested in the label column, the **Split Data** module is a quick solution.

Example of using the Split Data module

Suppose you imported a very large dataset from a CSV file. The dataset contains customer demographics. You want to create different models for customers in different countries, so you decide to split the data by using the value of the `Country-Region` column. Here are the steps you take to complete this task:

1. Add the **Split Data** module, and then specify an expression on the `Country-Region` field. The remainder of the data is available on the secondary output.
2. Add another instance of the **Split Data** module.
3. Repeat steps 1 and 2. Specify a different country in the expression for each iteration.

The Split Data module supports both *regular expressions*, for text data, and *relative expressions*, for numeric data.

The Split Data module also provides sophisticated functionality that you can use to divide specialized datasets. Use the functionality to create recommendation models, and to generate predictions.

Uses of the Partition and Sample module

- **Sampling.** Always use the [Partition and Sample](#) module. The module provides multiple customizable sampling methods, including several options for stratified sampling.
- **Assign cases to multiple groups.** Use the **Assign to Fold** or **Pick Fold** options in the [Partition and Sample](#) module.
- **Return only a subset of the data.** Use the [Partition and Sample](#) module. The module gives you the specified subset on the primary output. The remaining data is available on a secondary output.
- **Get only the top 2,000 rows of a dataset.** Use the [Partition and Sample](#) module. Select the **Head** option. This is particularly handy when you are testing a new experiment and want to run short trials of a workflow.

Example of using the Partition and Sample module

The [Partition and Sample](#) module can generate multiple partitions of the data, not just two. At the same time, it can perform various sampling operations.

For example, suppose you need to get just 10 percent of your data, while ensuring that the distribution of the target attribute is the same as in the source data. Here are the steps you take to complete this task:

1. Add the [Partition and Sample](#) module.
2. Choose the **Sampling** mode, and then specify **10%**.
3. Select the stratified sampling option, and then pick the column that contains the target attribute.

If you don't need to keep all the data, use the [Partition and Sample](#) module. The remaining data is still present in the workspace, but it doesn't need to be processed further as part of the experiment.

Related tasks

- Increase the number of rare cases in a sample, or rebalance the cases for a target value: Use the [SMOTE](#) module.
- Perform dimensionality reduction by finding the combination of features that best represents the data space: Use the [Principal Component Analysis](#) module.
- Create compact features based on an analysis of features and counts: Use the [Learning with Counts](#) module.
- Create a view or projection by using only the specified columns; remove or hide columns in a dataset: Use the [Select Columns in Dataset](#) and [Apply SQL Transformation](#) modules.
- Apply more complex data filters, groupings, or transformations: Use the [Execute R Script](#) and [Apply SQL Transformation](#) modules.

List of modules

This category includes the following modules:

- [Partition and Sample](#): Creates multiple partitions of a dataset based on sampling.
- [Split Data](#): Partitions the rows of a dataset into two distinct sets.

See also

- [Data Transformation](#)
- [A-Z module list](#)

Partition and Sample

3/10/2021 • 12 minutes to read • [Edit Online](#)

Creates multiple partitions of a dataset based on sampling

Category: [Data Transformation / Sample and Split](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Partition and Sample** module in Azure Machine Learning Studio (classic), to perform sampling on a dataset or to create partitions from your dataset.

Sampling is an important tool in machine learning because it lets you reduce the size of a dataset while maintaining the same ratio of values. This module supports several related tasks that are important in machine learning:

- Dividing your data into multiple subsections of the same size.

You might use the partitions for cross-validation, or to assign cases to random groups.

- Separating data into groups and then working with data from a specific group.

After randomly assigning cases to different groups, you might need to modify the features that are associated with only one group.

- Sampling.

You can extract a percentage of the data, apply random sampling, or choose a column to use for balancing the dataset and perform stratified sampling on its values.

- Creating a smaller dataset for testing.

If you have a lot of data, you might want to use only the first n rows while setting up the experiment, and then switch to using the full dataset when you build your model. You can also use sampling to create a smaller dataset for use in development.

How to configure Partition and Sample

This module supports multiple methods for dividing your data into partitions or for sampling. Choose the method first, and then set additional options required by the method.

- Get the top number of rows

Get TOP N rows from a dataset

Use this mode to get only the first n rows. This option is useful if you want to test an experiment on a small number of rows, and don't need the data to be balanced or sampled in any way.

1. Add the **Partition and Sample** module to your experiment in Studio (classic), and connect the dataset.

2. **Partition or sample mode:** Set this option to **Head**.
3. **Number of rows to select:** Type the number of rows to return.

The number of rows you specify must be a non-negative integer. If the number of selected rows is larger than the number of rows in the dataset, the entire dataset is returned.

4. Run the experiment.

The module outputs a single dataset containing only the specified number of rows. The rows are always read from the top of the dataset.

Create a sample of data

This option supports simple random sampling or stratified random sampling. This is useful if you want to create a smaller representative sample dataset for testing.

1. Add the **Partition and Sample** module to your experiment in Studio (classic), and connect the dataset.
2. **Partition or sample mode:** Set this to **Sampling**.

3. **Rate of sampling:** Type a value between 0 and 1. this value specifies the percentage of rows from the source dataset that should be included in the output dataset.

For example, if you want only half of the original dataset, type `0.5` to indicate that the sampling rate should be 50%.

The rows of the input dataset are shuffled and selectively put into the output dataset, according to the specified ratio.

4. **Random seed for sampling:** Optionally, type an integer to use as a seed value.

This option is important if you want the rows to be divided the same way every time. The default value is 0, meaning that a starting seed is generated based on the system clock. This can lead to slightly different results each time you run the experiment.

5. **Stratified split for sampling:** Select this option if it is important that the rows in the dataset should be divided evenly by some key column before sampling.

For **Stratification key column for sampling**, select a single *strata column* to use when dividing the dataset. The rows in the dataset are then divided as follows:

- All input rows are grouped (stratified) by the values in the specified strata column.
- Rows are shuffled within each group.
- Each group is selectively added to the output dataset to meet the specified ratio.

For more information about stratified sampling, see the [Technical notes](#) section.

6. Run the experiment.

With this option, the module outputs a single dataset that contains a representative sampling of the data.

The remaining, unsampled portion of the dataset is not output. However, you can create join on the datasets, using the [Apply SQL Transformation](#) module, to determine which rows were unused.

Split data into partitions

Use this option when you want to divide the dataset into subsets of the data. This option is also useful when you want to create a custom number of folds for cross-validation, or to split rows into several groups.

1. Add the **Partition and Sample** module to your experiment in Studio (classic), and connect the dataset.

2. For **Partition or sample mode**, select **Assign to Folds**.
 3. **Use replacement in the partitioning:** Select this option if you want the sampled row to be put back into the pool of rows for potential reuse. As a result, the same row might be assigned to several folds.

If you do not use replacement (the default option), the sampled row is not put back into the pool of rows for potential reuse. As a result, each row can be assigned to only one fold.
 4. **Randomized split:** Select this option if you want rows to be randomly assigned to folds.

If you do not select this option, rows are assigned to folds using the round-robin method.
 5. **Random seed:** Optionally, type an integer to use as the seed value. This option is important if you want the rows to be divided the same way every time. Otherwise, the default value of 0 means that a random starting seed will be used.
 6. **Specify the partitioner method:** Indicate how you want data to be apportioned to each partition, using these options:
 - **Partition evenly:** Use this option to place an equal number of rows in each partition. To specify the number of output partitions, type a whole number in the **Specify number of folds to split evenly into** text box.
 - **Partition with customized proportions:** Use this option to specify the size of each partition as a comma-separated list.

For example, if you want to create three partitions, with the first partition containing 50% of the data, and the remaining two partitions each containing 25% of the data, click the **List of proportions separated by comma** text box, and type these numbers: **.5, .25, .25**

The sum of all partition sizes must add up to exactly 1.

 - If you enter numbers that add up to **less than 1**, an extra partition is created to hold the remaining rows. For example, if you type the values **.2** and **.3**, a third partition is created that holds the remaining 50 percent of all rows.
 - If you enter numbers that add up to **more than 1**, an error is raised when you run the experiment.
 7. **Stratified split:** Select this option if you want the rows to be stratified when split, and then choose the **strata column**.

For more information about stratified sampling, see the [Technical notes](#) section.
 8. Run the experiment.

With this option, the module outputs multiple datasets, partitioned using the rules you specified.
- ### Use data from a predefined partition
- This option is used when you have divided a dataset into multiple partitions and now want to load each partition in turn for further analysis or processing.
1. Add the **Partition and Sample** module to the experiment in Studio (classic).
 2. Connect it to the output of a previous instance of **Partition and Sample**. That instance must have used the **Assign to Folds** option to generate some number of partitions.
 3. **Partition or sample mode:** Select **Pick Fold**.
 4. **Specify which fold to be sampled from:** Select a partition to use by typing its index. Partition indices are 1-based. For example, if you divided the dataset into three parts, the partitions would have the indices

1, 2, and 3.

If you type an invalid index value, a design-time error is raised: "Error 0018: Dataset contains invalid data."

In addition to grouping the dataset by folds, you can separate the dataset into two groups: a target fold, and everything else. To do this, type the index of a single fold, and then select the option, **Pick complement of the selected fold**, to get everything but the data in the specified fold.

5. If you are working with multiple partitions, you must add additional instances of the **Partition and Sample** module to handle each partition.

For example, let's say previously partitioned patients into five folds using age. To work with each individual fold, you need five copies of the **Partition and Sample** module, and in each, you select a different fold.

TIP

The sample experiment, [Split Partition and Sample](#), demonstrates this technique.

6. Run the experiment.

With this option, the module outputs a single dataset containing only the rows assigned to that fold.

NOTE

You cannot view the fold designations directly; they are present only in the metadata.

Examples

For examples of how this module is used, see the [Azure AI Gallery](#):

- [Cross Validation for Binary Classification](#): A 20% sampling rate is applied to create a smaller randomly sampled dataset. The original census dataset had over 30,000 rows; the sampled dataset has around 6500.
- [Cross Validation for Regression](#): The data is randomly and evenly assigned to five folds, with no stratification, and the results are used for cross-validation.
- [Split Partition and Sample](#): Demonstrates multiple ways to use partitioning and sampling. First, the **Assign to Folds** option is used to assign rows in the dataset to one of three evenly sized groups. Then, three more instances of **Partition and Sample** are added by using the **Pick Fold** mode to apply operations to subsets of the data
 - In the first fold (index of 1), rows are split randomly.
 - In the second fold (index of 2), rows are split by education.
 - In the third fold (index of 3), rows are split by age.

Technical notes

- The stratification column must be categorical with discrete values. If the column is not already categorical and you get an error, use [Edit Metadata](#) to change the column properties.
- The strata column you specify cannot contain continuous data: that is, numerical data with floating point values in each cell. Otherwise, the module cannot process the data and returns an error.

The reason is that any column used for stratification must have a finite set of possible values. If the specified strata column contains any floating point values, and the column is not of the type categorical, it potentially contains an infinite number of values.

- If the strata column contains Boolean values and you want them to be interpreted as categorical, you must use the [Edit Metadata](#) module to change the metadata label.
- If your strata column contains string or numerical data with too many unique values, the column is not a good candidate for stratified sampling.

More about stratified sampling

Stratified sampling ensures that subsets of the data have a representative sampling of the selected strata column. This technique is useful, for example, when you want to ensure that your training data contains the same distribution of age values that the test data has or vice versa. Or you might want to stratify a gender column in a health-care study to ensure that males and females are distributed evenly when the data is partitioned. Stratification ensures that the ratios of the selected values are preserved.

You specify values on which to separate the data by selecting a single column to serve as the **strata column**.

This module requires that the strata column is a categorical column. If you want to use a column of integer values for the strata, it is a best practice to assign a categorical type to this column. You can do this through the schema of the data before you add it to Azure Machine Learning Studio (classic), or you can update the column's metadata by using [Edit Metadata](#).

Columns with continuous data (that is, numerical data with floating point values in each cell) cannot be used as strata columns. If you get an error, you can use [Group Data into Bins](#) to bucket the values into discrete ranges, and then use [Edit Metadata](#) to guarantee that the column will be treated as categorical.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Dataset to be split

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Partition or sample mode	List	Sampling methods	Sampling	Select the partition or sampling mode
Use replacement in the partitioning	Any	Boolean	False	Indicate if the folds should be disjoint (default - no replacement) or overlapping (true - use replacement)
Randomized split	Any	Boolean	True	Indicate if the split is random
Random seed	Any	Integer	0	Specify a seed for the random number generator

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Specify the partitioner method	List	Partition methods	Partition evenly	Select Partition Evenly to partition into folds of equal size, or Partition with customized proportions to partition into folds of customized size
Specify number of folds to split evenly into	$>= 1$	Integer	5	Select a number of partitions to split into
Stratified split	List	True/False type	False	Indicate if the split is stratified
Stratification key column	Any	ColumnSelection		Contains the stratification key
List of proportions separated by comma	Any	String		List proportions, separated by commas
Stratified split for customized fold assignment	Any	True/False type	False	Indicate if the split is stratified for customized fold assignments
Stratification key column for customized fold assignment	Any	ColumnSelection		Contains the stratification key for customized fold assignments
Specify which fold to be sampled from	$>= 1$	Integer	1	Contains index of the fold to be sampled
Pick complement of the selected fold	Any	Boolean	False	Select the complement of the specified fold
Rate of sampling	Any	Float	0.01	Choose a sampling rate
Random seed for sampling	Any	Integer	0	Specify a seed for the random number generator for sampling
Stratified split for sampling	Any	True/False	False	Indicate if the split is stratified for sampling
Stratification key column for sampling	Any	ColumnSelection		Contains stratification key for sampling

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Number of rows to select	>=0	Integer	10	Select a maximum number of records that will be allowed to pass through to the next module

Outputs

NAME	TYPE	DESCRIPTION
oDataset	Data Table	Dataset resulting from the split

See also

[Sample and Split](#)

[Split Data](#)

[Edit Metadata](#)

[Group Data into Bins](#)

Split Data

3/10/2021 • 4 minutes to read • [Edit Online](#)

Partitions the rows of a dataset into two distinct sets

Category: [Data Transformation / Sample and Split](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This topic describes how to use the [Split Data](#) module in Azure Machine Learning Studio (classic), to divide a dataset into two distinct sets.

This module is particularly useful when you need to separate data into training and testing sets. You can customize the way that data is divided as well. Some options support randomization of data; others are tailored for a certain data type or model type.

How to configure Split Data

TIP

Before choosing the splitting mode, read all options to determine the type of split you need. If you change the splitting mode, all other options could be reset.

1. Add the **Split Data** module to your experiment in studio. You can find this module under **Data Transformation**, in the **Sample and Split** category.
2. **Splitting mode:** Choose one of the following modes, depending on the type of data you have, and how you want to divide it. Each splitting mode has different options. Click the following topics for detailed instructions and examples.
 - **Split Rows:** Use this option if you just want to divide the data into two parts. You can specify the percentage of data to put in each split, but by default, the data is divided 50-50.
You can also randomize the selection of rows in each group, and use stratified sampling. In stratified sampling, you must select a single column of data for which you want values to be apportioned equally among the two result datasets.
 - **Recommender Split:** Always choose this option if you are preparing data for use in a recommender system. It helps you divide data sets into training and testing groups while ensuring that important values such as user-item pairs or ratings are evenly divided among the groups.
 - **Regular Expression Split:** Choose this option when you want to divide your dataset by testing a single column for a value.

For example, if you are analyzing sentiment, you could check for the presence of a particular

product name in a text field, and then divide the dataset into rows with the target product name, and those without.

- **Relative Expression Split:** Use this option whenever you want to apply a condition to a number column. The number could be a date/time field, a column containing age or dollar amounts, or even a percentage. For example, you might want to divide your data set depending on the cost of the items, group people by age ranges, or separate data by a calendar date.

Requirements

- **Split Data** can create a maximum of two datasets sets at a time, and those sets must be exclusive.

Therefore, if you have a complex split with multiple conditions and outputs, you might need to chain together multiple **Split Data** modules.

Alternatively, you can use a CASE statement and the [Apply SQL Transformation](#) module.

- This module doesn't delete data or remove it from the dataset; it just divides the data as specified among the first and second outputs of the module.
- Splitting data for a recommender system entails some additional requirements. In general, the dataset can only consist of user-item pairs or user-item-rating triples. Therefore, the **Split Data** module cannot work on datasets that have more than three columns, to avoid confusion with feature-type data. If your dataset contains too many columns, you might get this error:

Error 0022: Number of selected columns in input dataset does not equal to x

As a workaround, you can use [Select Columns in Dataset](#) to remove some columns, and then add the columns later using [Add Columns](#). Alternatively, if your dataset has many features that you want to use in the model, divide the dataset using a different option, and train the model using [Train Model](#) rather than [Train Matchbox Recommender](#).

Examples

For examples of how the **Split Data** module is used, see the [Azure AI Gallery](#):

- [Cross Validation for Binary Classification: Adult Dataset](#): A 20% sampling rate is applied to create a smaller randomly sampled dataset. (The original census dataset had over 30,000 rows; the training dataset has around 6500). The dataset is cleaned for missing values and then passed to five different models for training and cross-validation.

Technical notes

The following requirements apply to all uses of **Split Data**:

- The input dataset must contain at least two rows, or an error is raised.
- If you use the option to specify the desired number of rows, the specified number must be a positive integer, and the number must be less than the total number of rows in the dataset.
- If you specify a number as a percentage, or if you use a string that contains the "%" character, the value is interpreted as a percentage. All percentage values must be within the range (0, 100), not including the values 0 and 100.
- If you specify a number or percentage that is a floating point number less than one, and you do not use the percent symbol (%), the number is interpreted as a proportional value.
- If you use the option for a stratified split, the output datasets can be further divided by subgroups, by selecting a strata column.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Dataset to split

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DESCRIPTION	DEFAULT
Splitting mode	Split mode	Split Rows, Recommender Split, Regular Expression, or Relative Expression	Required	Split Rows	Choose the method for splitting the dataset

Outputs

NAME	TYPE	DESCRIPTION
Results dataset1	Data Table	Dataset that contains selected rows
Results dataset2	Data Table	Dataset that contains all other rows

See also

[Sample and Split](#)

[Partition and Sample](#)

[A-Z Module List](#)

Split Data using Split Rows

3/10/2021 • 4 minutes to read • [Edit Online](#)

This article describes how to use the **Split Rows** option in the [Split Data](#) module of Azure Machine Learning Studio (classic). This option is particularly useful when you need to divide datasets used for training and testing, either randomly or by some criteria.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The **Split Rows** option supports both random and stratified splits. For example, you can specify a 70-30 split, or a 10-90 split with your target variable equally represented in both datasets.

For general information about data partitioning for machine learning experiments, see [Split Data](#) and [Partition and Split](#).

Related tasks

Other options in the **Split Data** module support different ways to divide the data:

- [Split data using regular expressions](#): Apply a regular expression to a single text column, and divide the dataset based on the results.
- [Split data using relative expressions](#): Apply an expression to a numeric column, and divide the dataset based on the results
- [Split recommender datasets](#): Divide datasets that are used in recommendation models. The dataset should have three columns: items, users, and ratings.

Divide a dataset into two groups

1. Add the [Split Data](#) module to your experiment in Studio (classic), and connect the dataset you want to split.
2. For **Splitting mode**, choose **Split rows**.
3. **Fraction of rows in the first output dataset**. Use this option to determine how many rows go into the first (left-hand) output. All other rows will go to the second (right-hand) output.

The ratio represents the percentage of rows sent to the first output dataset, so you must type a decimal number between 0 and 1.

For example, if you type 0.75 as the value, the dataset would be split by using a 75:25 ratio, with 75% of the rows sent to the first output dataset, and 25% sent to the second output dataset.

4. Select the **Randomized split** option if you want to randomize selection of data into the two groups. This is the preferred option when creating training and test datasets.
5. **Random Seed**: Type a non-negative integer value to initialize the pseudorandom sequence of instances to be used. This default seed is used in all modules that generate random numbers.

Specifying a seed makes the results generally reproducible. If you need to repeat the results of a split operation, you should specify a seed for the random number generator. Otherwise the random seed is set by default to 0, which means the initial seed value is obtained from the system clock. As a result, the distribution of data might be slightly different each time you perform a split.

6. **Stratified split:** Set this option to **True** to ensure that the two output datasets contain a representative sample of the values in the *strata column* or *stratification key column*.

With stratified sampling, the data is divided such that each output dataset gets roughly the same percentage of each target value. For example, you might want to ensure that your training and testing sets are roughly balanced with regard to the outcome, or with regard to some other column such as gender.

7. Run the experiment, or right-click the module and select **Run selected**.

Examples

The following examples demonstrate how to perform simple splits using **Split Rows** mode.

Split into two equal parts

Add the [Split Data](#) module after the dataset without no other changes. By default, the module splits the dataset in two equal parts. For data with an odd number of rows, the second output gets the remainder.

Split into thirds

Assume that you want to split a dataset into two parts, with a third of the data used for training and the remainder for testing or additional splits.

To do this, add a [Split Data](#) module, and set the **Fraction of rows in the first output** to 0.33. The second output contains the remaining two-thirds.

To divide the second output into equal parts, add another instance of the [Split Data](#) module, and this time use the default for a 50-50 split.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

- This module requires that the dataset contain at least two rows; otherwise, an error is raised.
- If you use the option to specify the desired number of rows, the specified number must be a positive integer, and the number must be less than the total number of rows in the dataset.
- All percentage values must be within the range 0 and 1.
- If you specify a number or percentage as a floating point number less than one, and you do not use the percent symbol (%), the number is interpreted as a proportional value.

Additional requirements for stratified sampling

- The strata column can contain only nominal or categorical data. If the column contains continuous numeric data, an error message is raised.
- A column with too many unique values is not a good candidate for stratification. You might try collapsing some categories or grouping values beforehand.

See also

Sample and Split Partition and Sample

Split Data using Recommender Split

3/10/2021 • 6 minutes to read • [Edit Online](#)

This article describes how to use the **Recommender Split** option in the [Split Data](#) module of Azure Machine Learning Studio (classic). This option is useful when you need to prepare training and testing datasets for use with a recommendation model. Not only do these models require a specific format, but it can be very difficult to divide up ratings, users, and items in a balanced way without special tools.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The **Recommender split** option makes this process easier by asking for the type of recommendation model you are working with: for example, are you recommending items, suggesting a rating, or finding related users? It then divides the dataset by criteria you specify, such as how to handle cold users or cold items.

When you split the datasets, the module returns two datasets, one intended for training and the other for testing or model evaluation. If the input dataset contains any extra data per instance (such as ratings), it is preserved in the output.

For general information about data partitioning for machine learning experiments, see

Related tasks

Other options in the **Split Data** module support different ways to divide the data:

- [Split data using regular expressions](#): Apply a regular expression to a single text column, and divide the dataset based on the results
- [Split recommender datasets](#): Divide datasets + [Split data using relative expressions](#): Apply an expression to numeric data.
- [Split by percentage of dataset](#)

Divide a dataset used by a recommendation model

The **Recommender Split** option is provided specifically for data used to train recommendation systems.

Before you use this option, make sure that your data is in a compatible format. The recommender splitter works under the assumption the dataset consists only of **user-item pairs** or **user-item-rating triples**. For details, see [Input data requirements](#) in this article.

1. Add the [Split Data](#) module to your experiment, and connect it as input to the dataset you want to split.
2. For **Splitting mode**, select **Recommender split**.
3. Set the following options to control how values are divided. Specify a percentage represented as a number between 0 and 1.
 - **Fraction of training only users**: Specify the fraction of users that should be assigned only to the training data set. This means the rows would never be used to test the model.

- **Fraction of test user ratings for training:** Specify that some portion of the user ratings you have collected can be used for training.
 - **Fraction of cold users:** Cold users are users that the system has not previously encountered. Typically, because the system has no information on these users, they are valuable for training, but predictions might be less accurate.
 - **Fraction of cold items:** Cold items are items that the system has not previously encountered. Because the system has no information about these items, they are valuable for training, but predictions might be less accurate.
 - **Fraction of ignored users:** This option allows the recommender to ignore some users, which lets you train the model on a subset of data. This might be useful for performance reasons. You specify the percentage of users that should be ignored.
 - **Fraction of ignored items:** The recommender splitter can ignore some items and train the model on a subset of data. This might be useful for performance reasons. You specify the percentage of items to ignore.
4. **Remove occasionally produced cold items:** This option is typically set to zero, to ensure that all entities in the test set are included in the training set.
- An item is said to be "occasionally cold" if it is covered only by the test set and it wasn't explicitly chosen as cold. Such items can be produced by steps (4) and (6) in the algorithm described in the [How Recommender Data is Split](#) section.
5. **Random seed for recommender:** Specify a seed value if you want to split the data the same way every time. Otherwise, by default the input data is randomly split, using a system clock value as the seed.
6. Run the experiment.

Examples

For examples of how to divide a set of ratings and features used for training or testing a recommendation model, we recommend that you review the walkthrough provided with this sample experiment in the [Azure AI Gallery: Movie Recommendation](#)

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Requirements for input data

The recommender splitter works under the assumption the dataset consists only of user-item pairs or user-item-rating triples. Therefore, the [Split Data](#) module cannot work on datasets that have more than three columns, to avoid confusion with feature-type data.

If your dataset contains too many columns, you might get this error:

Error 0022: Number of selected columns in input dataset does not equal to x

As a workaround, you can use [Select Columns in Dataset](#) to remove some columns. You can always add the columns back later, by using the [Add Columns](#) module.

Alternatively, if your dataset has many features that you want to use in the model, divide the dataset using a different option, and train the model using [Train Model](#) rather than [Train Matchbox Recommender](#).

For detailed information about the supported data formats, see [Train Matchbox Recommender](#).

Usage tips

- An error is raised if the dataset does not contain at least two rows.
- If you specify a number as a percentage, or if you use a string that contains the "%" character, the value is interpreted as a percentage.

All percentage values must be within the range (0, 100), not including the values 0 and 100.

- If you specify a number or percentage that is a floating point number less than one, and you do not use the percent symbol (%), the number is interpreted as a proportional value.

Implementation details

The following algorithm is used when splitting data into training and test sets for use with a recommendation model:

1. The requested fraction of ignored items is removed with all associated observations.
2. The requested fraction of cold items is moved to the test set with all associated observations.
3. The requested fraction of ignored users that remain after the first two steps is removed with all associated observations.
4. The requested fraction of cold users that remain after the first two steps is moved to the test set with all associated observations.
5. The requested fraction of training-only users that remain after the first two steps is moved to the training set with all associated observations.
6. For each user that remains after all the previous steps, the requested fraction of test user ratings for training is moved to the training set, and the remainder is moved to the test set.

At least one observation is always moved to the training set for each user.

7. If requested, instances that are associated with the occasionally produced cold items can be removed from the test set.

An item is said to be "occasionally cold" if it is covered only by the test set, and it wasn't explicitly chosen as cold. Such items can be produced by steps (4) and (6).

The anticipated use of this option is that the requested number of cold users and items is set to zero. This ensures that all entities in the test set are included in the training set.

See also

[Split Data Partition and Split](#)

Split Data using Regular Expression

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes how to use the **Regular Expression Split** option in the [Split Data](#) module of Azure Machine Learning Studio (classic). This option is useful when you need to apply a filter criteria to a text column. For example, you might divide your dataset by whether a particular product is mentioned.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

You can use a **regular expression split** on a single text column. You define a regular expression that includes the text column name, and then set conditions that apply to the column, such as "begins with", ""contains", or "does not contain".

For general information about data partitioning for machine learning experiments, see [Split Data](#) and [Partition and Split](#).

Related tasks

Other options in the **Split Data** module:

- [Split data using relative expressions](#): Apply an expression to numeric data.
- [Split recommender datasets](#): Divide datasets that are used in recommendation models. The dataset should have three columns: items, users, and ratings
- [Split by percentage of dataset](#)

Use a regular expression to divide a dataset

1. Add the [Split Data](#) module to your experiment, and connect it as input to the dataset you want to split.
2. For **Splitting mode**, select **Regular expression split**.
3. In the **Regular expression** box, type a valid regular expression. Some examples are provided [here](#).

The regular expression is applied only to the specified column, which must be a string data type.

For help composing regular expressions, see the [Regular Expression Language - Quick Reference](#).

4. Run the experiment, or right-click the module and select **Run selected**.

Based on the regular expression you provide, the dataset is divided into two sets of rows: rows with values that match the expression and all remaining rows.

Examples

The following examples demonstrate how to divide a dataset using the **Regular Expression** option.

Single whole word

This example puts into the first dataset all rows that contain the text `Gryphon` in the column `Text`, and puts other rows into the second output of **Split Data**:

```
\"Text\" Gryphon
```

Substring

This example looks for the specified string in any position within the second column of the dataset, denoted here by the index value of 1. The match is case-sensitive.

```
(\1) ^[a-f]
```

The first result dataset contains all rows where the index column begins with one of these characters: `a`, `b`, `c`, `d`, `e`, `f`. All other rows are directed to the second output.

String match on IP addresses

This example divides some server log data into two categories for analysis: connections behind the firewall and connections with IP addresses outside the firewall. The regular expression is applied to the `IP_Address` field (a **string** data type).

```
(\IP_Address) ^[10]
```

The first output contains all addresses that begin with `10`.

See also

[Sample and Split](#)

[Partition and Sample](#)

Split a dataset using a relative expression

3/10/2021 • 3 minutes to read • [Edit Online](#)

This article describes how to use the **Relative Expression Split** option in the [Split Data](#) module of Azure Machine Learning Studio (classic). This option is helpful when you need to divide a dataset into training and testing datasets using a numerical expression. For example:

- Age greater than 40 vs. 40 or younger
- Test score of 60 or higher vs. less than 60
- Rank value of 1 vs. all other values

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

To divide your data, you choose a single numeric column in your data, and define an expression to use in evaluating each row. The *relative expression* must include the column name, the value, and an operator such as greater than and less than, equal and not equals.

This option divides the dataset into **two** groups.

For general information about data partitioning for machine learning experiments, see [Split Data](#) and [Partition and Split](#).

Related tasks

Other options in the **Split Data** module:

- [Split data using regular expressions](#): Apply a regular expression to a single text column, and divide the dataset based on the results
- [Split recommender datasets](#): Divide datasets that are used in recommendation models. The dataset should have three columns: items, users, and ratings
- [Split by percentage of dataset](#)

Use a relative expression to divide a dataset

1. Add the [Split Data](#) module to your experiment in Studio, and connect it as input to the dataset you want to split.
2. For **Splitting mode**, select **relative expression split**.
3. In the **Relational expression** text box, type an expression that performs a numeric comparison operation, on a single column:
 - The column contains numbers of any numeric data type, including date/time data types.
 - The expression can reference a maximum of one column name.
 - Use the ampersand character (&) for the AND operation and use the pipe character (|) for the OR

operation.

- The following operators are supported: <, >, <=, >=, ==, !=
- You cannot group operations by using (and).

For ideas, see the [Examples](#) section.

4. Run the experiment, or right-click the module and select **Run selected**.

The expression divides the dataset into two sets of rows: rows with values that meet the condition, and all remaining rows.

If you need to perform additional split operations, you can either add a second instance of **Split Data*, or use the [Apply SQL Transformation](#) module and define a CASE statement.

Examples of relative expressions

The following examples demonstrate how to divide a dataset using the **Relative Expression** option in the **Split Data** module:

Using calendar year

A common scenario is to divide a dataset by years. The following expression selects all rows where the values in the column `Year` are greater than `2010`.

```
\\"Year" > 2010
```

The date expression must account for all date parts that are included in the data column, and the format of dates in the data column must be consistent.

For example, in a date column using the format `mmddyyyy`, the expression should be something like this:

```
\\"Date" > 1/1/2010
```

Using column indices

The following expression demonstrates how you can use the column index to select all rows in the first column of the dataset that contain values less than or equal to 30, but not equal to 20.

```
(\0)<=30 & !=20
```

Compound operation on time values using multiple splits

Suppose you want to split a table of log data, to group queries that run too long. You could use the following relative expression on the column, `Elapsed`, to get the queries that ran over 1 minute.

```
\\"Elapsed" >00:01:00
```

To get the queries with response times under one minute but more than 30 seconds, add another instance of [Split Data](#) on the right-hand output, and use an expression like this:

```
\\"Elapsed" <:00:01:00 & >00:00:30
```

Split dataset on date values

The following relative expression divides the dataset by using the date values in the column `dt1`.

```
\"dt1" > 10-08-2015
```

Rows with a date greater than 10-08-2015 are added to the first (left) output dataset.

Rows with a date of 10-08-2015 or earlier are added to the second (right) output dataset.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Restrictions

The following restrictions apply to relative expressions on a dataset:

- Relative expressions can be applied only to numeric data types and date/time data types.
- Relative expressions can reference a maximum of one column name.
- Use the ampersand character (&) for the AND operation and the pipe character (|) for the OR operation.
- The following operators are allowed for relative expressions: `<`, `>`, `<=`, `>=`, `==`, `!=`
- Grouping operations with parentheses is not supported.

See also

[Sample and Split](#)

[Partition and Sample](#)

Data Transformation - Scale and Reduce

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that can help you work with numerical data. For machine learning, common data tasks include clipping, binning, and normalizing numerical values. Other modules support dimensionality reduction.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Modeling numerical data

Tasks such as normalizing, binning, or redistributing numerical variables are an important part of data preparation for machine learning. The modules in this group support the following data preparation tasks:

- Grouping data into bins of varying sizes or distributions.
- Removing outliers or changing their values.
- Normalizing a set of numeric values into a specific range.
- Creating a compact set of feature columns from a high-dimension dataset.

Related tasks

- Select relevant and useful features to use in building the model: Use the [Feature Selection](#) or [Fisher Linear Discriminant Analysis](#) modules.
- Select features based on counts of the values: Use the [Learning with Counts](#) module.
- Remove or replace missing values: Use the [Clean Missing Data](#) module.
- Replace categorical values with numerical values that are derived from calculations: Use the [Replace Discrete Values](#) module.
- Compute a probability distribution for discrete or numerical columns: Use the [Evaluate Probability Function](#) module.
- Filter and transform digital signals and waveforms: Use the [Filter](#) module.

List of modules

This **Data Transformation - Scale and Reduce** category includes the following modules:

- [Clip Values](#): Detects outliers, and then clips or replaces their values.
- [Group Data into Bins](#): Puts numerical data into bins.
- [Normalize Data](#): Rescales numeric data to constrain dataset values to a standard range.
- [Principal Component Analysis](#): Computes a set of features that have reduced dimensionality for more efficient learning.

See also

- [Manipulation](#)

- [Sample and Split](#)
- [Filter](#)
- [Learning with Counts](#)
- [Feature Selection](#)
- [Module categories and descriptions](#)
- [A-Z module list](#)

Clip Values

3/10/2021 • 8 minutes to read • [Edit Online](#)

Detects outliers and clips or replaces their values

Category: [Data Transformation / Scale and Reduce](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Clip Values** module in Azure Machine Learning Studio (classic), to identify and optionally replace data values that are above or below a specified threshold. This is useful when you want to remove outliers or replace them with a mean, a constant, or other substitute value.

You connect the module to a dataset that has the numbers you want to clip, choose the columns to work with, and then set a threshold or range of values, and a replacement method. The module can output either just the results, or the changed values appended to the original dataset.

How to configure Clip Values

Before you begin, identify the columns you want to clip, and the method to use. We recommend that you test any clipping method on a small subset of data first.

The module applies the same criteria and replacement method to **all** columns that you include in the selection. Therefore, be sure to exclude columns that you don't want to change.

If you need to apply clipping methods or different criteria to some columns, you must use a new instance of **Clip Values** for each set of similar columns.

1. Add the **Clip Values** module to your experiment and connect it to the dataset you want to modify. You can find this module under **Data Transformation**, in the **Scale and Reduce** category.
2. In **List of columns**, use the Column Selector to choose the columns to which **Clip Values** will be applied.
3. For **Set of thresholds**, choose one of the following options from the dropdown list. These options determine how you set the upper and lower boundaries for acceptable values vs. values that must be clipped.
 - **ClipPeaks**: When you clip values by peaks, you specify only an upper boundary. Values greater than that boundary value are replaced or removed.
 - **ClipSubpeaks**: When you clip values by sub-peaks, you specify only a lower boundary. Values that are less than that boundary value are replaced or removed.
 - **ClipPeaksAndSubpeaks**: When you clip values by peaks and sub-peaks, you can specify both the upper and lower boundaries. Values that are outside that range are replaced or removed. Values

that match the boundary values are not changed.

4. Depending on your selection in the preceding step, you can set the following threshold values:

- **Lower threshold:** Displayed only if you choose **ClipSubPeaks**
- **Upper threshold:** Displayed only if you choose **ClipPeaks**
- **Threshold:** Displayed only if you choose **ClipPeaksAndSubPeaks**

For each threshold type, choose either **Constant** or **Percentile**.

5. If you select **Constant**, type the maximum or minimum value in the text box. For example, assume that you know the value 999 was used as a placeholder value. You could choose **Constant** for the upper threshold, and type 999 in **Constant value of upper threshold**.

6. If you choose **Percentile**, you constrain the column values to a percentile range.

For example, assume you want to keep only the values in the 10-80 percentile range, and replace all others. You would choose **Percentile**, and then type 10 for **Percentile value of lower threshold**, and type 80 for **Percentile value of upper threshold**.

See the section on [percentiles](#) for some examples of how to use percentile ranges.

7. Define a substitute value.

Numbers that exactly match the boundaries you just specified are considered to be inside the allowed range of values, and thus are not replaced or removed. All numbers that fall outside the specified range are replaced with the substitute value.

- **Substitute value for peaks:** Defines the value to substitute for all column values that are greater than the specified threshold.
- **Substitute value for subpeaks:** Defines the value to use as a substitute for all column values that are less than the specified threshold.
- If you use the **ClipPeaksAndSubpeaks** option, you can specify separate replacement values for the upper and lower clipped values.

The following replacement values are supported:

- **Threshold:** Replaces clipped values with the specified threshold value.
- **Mean:** Replaces clipped values with the mean of the column values. The mean is computed before values are clipped.
- **Median:** Replaces clipped values with the median of the column values. The median is computed before values are clipped.
- **Missing.** Replaces clipped values with the missing (empty) value.

8. **Add indicator columns:** Select this option if you want to generate a new column that tells you whether or not the specified clipping operation applied to the data in that row. This option is particularly handy when you are testing a new set of clipping and substitution values.

9. **Overwrite flag:** Indicate how you want the new values to be generated. By default, **Clip Values** constructs a new column with the peak values clipped to the desired threshold. New values overwrite the original column.

To keep the original column and add a new column with the clipped values, deselect this option.

10. Run the experiment.

Right-click the output of the **Clip Values** module and select **Visualize** to review the values and make sure the clipping operation met your expectations.

Examples

To see how this module is used in machine learning experiments, see the [Azure AI Gallery](#):

- [Forest Fire outliers](#): This example from the EdX course in data science demonstrates clipping methods using the Forest Fires sample dataset.

Clipping using percentiles

To understand how clipping by percentiles works, consider a dataset with 10 rows, which have one instance each of the values 1-10.

- If you are using percentile as the upper threshold, at the value for the 90th percentile, 90 percent of all values in the dataset must be less than that value.
- If you are using percentile as the lower threshold, at the value for the 10th percentile, 10 percent of all values in the dataset must be less than that value.

1. For **Set of thresholds**, choose **ClipPeaksAndSubPeaks**.
2. For **Upper threshold**, choose **Percentile**, and for **Percentile number**, type 90.
3. For **Upper substitute value**, choose **Missing Value**.
4. For **Lower threshold**, choose **Percentile**, and for **Percentile number**, type 10.
5. For **Lower substitute value**, choose **Missing Value**.
6. Deselect the option **Overwrite flag**, and select the option, **Add indicator column**.

Now try the same experiment using 60 as the upper percentile threshold and 30 as the lower percentile threshold, and use the threshold value as the replacement value. The following table compares these two results:

1. Replace with missing; Upper threshold = 90; Lower threshold = 10
2. Replace with threshold; Upper percentile = 60; Lower percentile = 30

ORIGINAL DATA	REPLACE WITH MISSING	REPLACE WITH THRESHOLD
1	TRUE	4, TRUE
2	TRUE	4, TRUE
3	3, FALSE	4, TRUE
4	4, FALSE	4, TRUE
5	5, FALSE	5, FALSE
6	6, FALSE	6, FALSE
7	7, FALSE	7, TRUE
8	8, FALSE	7, TRUE
9	9, FALSE	7, TRUE
10	TRUE	7, TRUE

Technical notes

- You can use **Clip Values** only on columns containing numbers or date/time values.

- If you include columns that have text or categorical data, the columns will be skipped.
- Missing values are ignored when the mean or median value is computed for a column.
- **Clip Values** does not support ordinal data.
- Missing values are not altered when they are propagated to the output dataset. The column indicating clipped values always contains FALSE for missing values.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Add indicator columns	TRUE/FALSE	Boolean	FALSE	Whether to add indicator for clipping of a value is done
Constant value for lower threshold	any	Float	-1	Value below which the subpeaks will be clipped
Constant value for upper threshold	any	Float	1	Value above which the peaks will be clipped
Constant value of lower threshold	any	Float	-1	Value below which the subpeaks are clipped
Constant value of upper threshold	>=1	Float	1	Value above which the peaks are clipped
List of columns		ColumnSelection		List of columns to clip
Lower substitute value	Threshold Mean Median Missing	SubstituteValues	Threshold	The value used for clipping subpeaks
Lower threshold	Constant Percentile	Threshold Mode	Constant	Value below which the subpeaks will be clipped mode
Overwrite flag	TRUE/FALSE	Boolean	TRUE	Whether clipped data column(s) must overwrite input data column(s)

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Percentile number for lower threshold	[1;99]	Integer	1	Percentile number below which the subpeaks will be clipped
Percentile number for upper threshold	[1;99]	Integer	99	Percentile number above which the peaks will be clipped
Percentile number of lower threshold	[1;99]	Integer	1	Percentile number below which the subpeaks are clipped
Percentile number of upper threshold	[1;99]	Integer	99	Percentile number above which the peaks are clipped
Set of thresholds	ClipPeaks ClipSubPeaks ClipPeaksAndSubPeaks	Threshold Set	ClipPeaks	Specifies type of threshold to use
Substitute value for peaks	Threshold Mean Median Missing	SubstituteValues	Threshold	The value used during clipping peaks
Substitute value for subpeaks	Threshold Mean Median Missing	SubstituteValues	Threshold	The value used during clipping subpeaks
Threshold	Constant Percentile	Threshold Mode	Constant	Value above and below which the peaks will be clipped mode
Upper substitute value	Threshold Mean Median Missing	Threshold	Threshold	The value used for clipping peaks
Upper threshold	Constant Percentile	Threshold Mode	Constant	Value above which the peaks will be clipped mode

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with clipped columns

Exceptions

EXCEPTION	DESCRIPTION
Error 0011	Exception occurs if passed column set argument does not apply to any of dataset columns.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Scale and Reduce](#)

[A-Z Module List](#)

Group Data into Bins

3/10/2021 • 12 minutes to read • [Edit Online](#)

Puts numerical data into bins

Category: [Scale and Reduce](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Group Data into Bins](#) module in Azure Machine Learning Studio (classic), to group numbers or change the distribution of continuous data.

The [Group Data into Bins](#) module supports multiple options for binning data. You can customize how the bin edges are set and how values are apportioned into the bins. For example, you can:

- Manually type a series of values to serve as the bin boundaries.
- Calculate entropy scores to determine information values for each range, to optimize the bins in the predictive model. + Assign values to bins by using *quantiles*, or percentile ranks.
- Control the number of values in each bin can also be controlled.
- Force an even distribution of values into the bins.

More about binning and grouping

Binning or grouping data (sometimes called *quantization*) is an important tool in preparing numerical data for machine learning, and is useful in scenarios like these:

- A column of continuous numbers has too many unique values to model effectively, so you automatically or manually assign the values to groups, to create a smaller set of discrete ranges.

For example, you could use entropy scores generated by [Group Data into Bins](#) to identify the optimal groupings of data values, and use those groups as features in your model.

- Replace a column of numbers with categorical values that represent specific ranges.

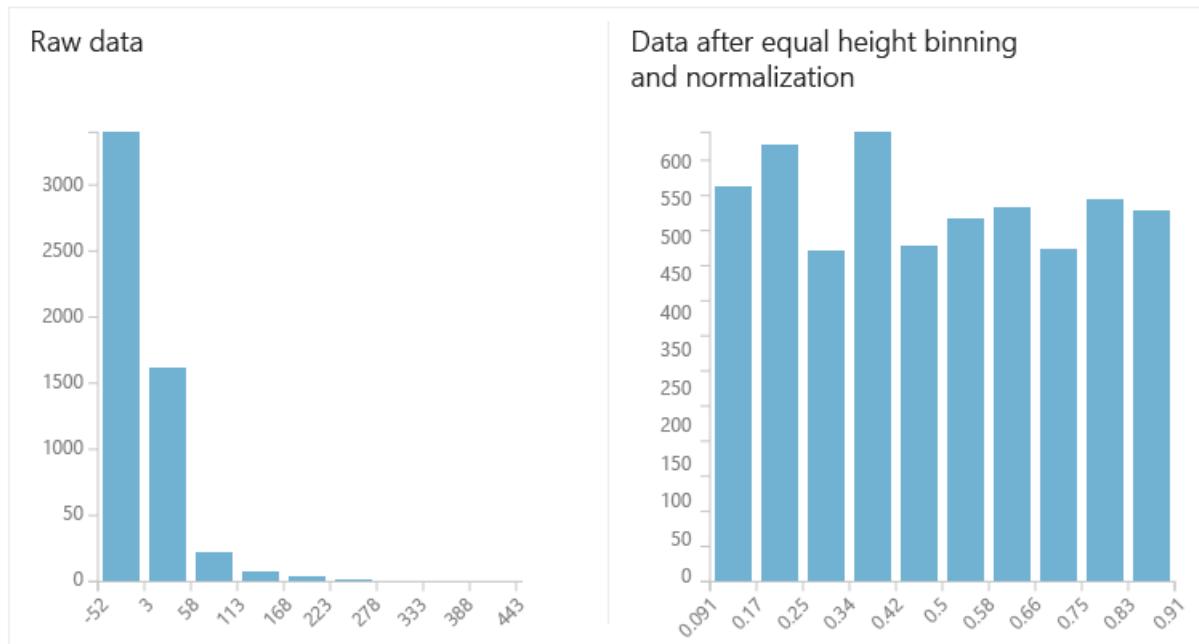
For example, you might want to group values in an age column by specifying custom ranges, such as 1-15, 16-22, 23-30, and so forth for user demographics.

- A dataset has a few extreme values, all well outside the expected range, and these values have an outsized influence on the trained model. To mitigate the bias in the model, you might transform the data to a uniform distribution, using the quantiles (or equal-height) method.

With this method, the [Group Data into Bins](#) module determines the ideal bin locations and bin widths to ensure that approximately the same number of samples fall into each bin. Then, depending on the normalization method you choose, the values in the bins are either transformed either to percentiles or mapped to a bin number.

Examples of binning

The following diagram shows the distribution of numeric values before and after binning with the **quantiles** method. Notice that compared to the raw data at left, the data has been binned and transformed to a unit-normal scale.



Another approach to binning is demonstrated in the [Breast cancer detection](#) sample, in which **Group Data into Bins** is used to assign patients to various control and test groups, to guarantee that each group has an equal number of patients.

Because there are so many ways to group data, all customizable, we recommend that you experiment with different methods and values. The [Examples](#) section contains links to sample experiments that demonstrate how to use the different binning algorithms.

How to configure Group Data into Bins

1. Add the **Group Data Into Bins** module to your experiment in Studio (classic). You can find this module in the category **Data Transformation**, under **Scale and Reduce**.
2. Connect the dataset that has numerical data to bin. Quantization can be applied only to columns containing numeric data.
If the dataset contains non-numeric columns, use the [Select Columns in Dataset](#) module to select a subset of columns to work with.
3. Specify the binning mode. The binning mode determines other parameters so be sure to select the **Binning mode** option first! The following types of binning are supported:

Entropy MDL: This method requires that you select the column you want to predict and the column or columns that you want to group into bins. It then makes a pass over the data and attempts to determine the number of bins that minimizes the entropy. In other words, it chooses a number of bins that allows the data column to best predict the target column. It then returns the bin number associated with each row of your data in a column named `<colname>quantized`.

If the **Entropy MDL** method cannot find a way to initially bin the data to make a good prediction, it assigns all data to a uniform bin. This does not mean that the column is not a good predictor. In this case, you can use other methods to find the number of bins that would minimize entropy, and make the data a better predictor.

This method does not return the actual entropy scores.

Quantiles: The quantile method assigns values to bins based on percentile ranks. Quantiles is also known as equal height binning.

Equal Width: With this option, you must specify the total number of bins. The values from the data column are placed in the bins such that each bin has the same interval between starting and end values. As a result, some bins might have more values if data is clumped around a certain point.

Custom Edges: You can specify the values that begin each bin. The edge value is always the lower boundary of the bin. For example, assume you want to group values into two bins, one with values greater than 0, and one with values less than or equal to 0. In this case, for bin edges, you would type 0 in **Comma-separated list of bin edges**. The output of the module would be 1 and 2, indicating the bin index for each row value.

Equal Width with Custom Start and Stop: This method is like the **Equal Width** option, but you can specify both lower and upper bin boundaries.

4. **Number of bins:** If you are using the **Entropy MDL**, **Quantiles**, and **Equal Width** binning modes, use this option to specify how many bins, or *quantiles*, that you want to create.

5. For **Columns to bin**, use the Column Selector to choose the columns that have the values you want to bin. Columns must be a numeric data type.

The same binning rule is applied to all applicable columns that you choose. Therefore, if you need to bin some columns by using a different method, use a separate instance of **Group Data into Bins** for each set of columns.

WARNING

If you choose a column that is not an allowed type, a run-time error is generated. The module returns an error as soon as it finds any column of a disallowed type. If you get an error, review all selected columns. The error does not list all invalid columns.

6. For **Output mode**, indicate how you want to output the quantized values.

- **Append:** Creates a new column with the binned values and appends that to the input table.
- **Inplace:** Replaces the original values with the new values in the dataset.
- **ResultOnly:** Returns just the result columns.

7. If you select the **Quantiles** binning mode, use the **Quantile normalization** option to determine how values are normalized prior to sorting into quantiles. Note that normalizing values transforms the values, but does not affect the final number of bins. For an example, see [Effects of Different Normalization Methods](#).

The following normalization types are supported:

- **Percent:** Values are normalized within the range [0,100]
 - **PQuantile:** Values are normalized within the range [0,1]
 - **QuantileIndex:** Values are normalized within the range [1,number of bins]
8. If you choose the **Custom Edges** option, type a comma-separated list of numbers to use as *bin edges* in the + **Comma-separated list of bin edges** text box. The values mark the point that divides bins. Therefore, if you type one bin edge value, two bins will be generated; if you type two bin edge values, three bins will be generated, and so forth.

The values must be sorted in order that the bins are created, from lowest to highest.

9. If you use the option, **Equal Width With Custom Start And Stop**, you must specify the boundaries of the bins.

Define the lower boundary of the first bin by typing a value in the **First edge position** text box.

Define the lower boundary of the last bin by typing a value in the **Last edge position** text box.

10. **Tag columns as categorical:** Select this option to automatically add a metadata flag to the column of binned values. The metadata flag indicates that the quantized columns should be handled as categorical variables.

11. Run the experiment, or select this module and click **Run selected**.

Results

The [Group Data into Bins](#) module returns a dataset in which each element has been binned according to the specified mode.

It also returns a **Binning transformation**, which is a function that can be passed to the [Apply Transformation](#) module to bin new samples of data using the same binning mode and parameters.

To see how well the binning method functions as a predictor, you can click the dataset output from [Group Data to Bins](#), and compare the label column to the binned column. If the grouping to bins is predictive, the values in the cross-tab matrix should concentrate in a few cells.

TIP

Remember, if you use binning on your training data, you must use the same binning method on data that you use for testing and prediction. This includes the binning method, bin locations, and bin widths.

To ensure that data is always transformed by using the same binning method, we recommend that you save useful data transformations, and then apply them to other datasets, by using the [Apply Transformation](#) module.

Examples

For examples of how quantization is applied in machine learning scenarios, see the [Azure AI Gallery](#):

- [Breast cancer detection](#): In this sample, binning is used to divide patients into equal groups by using the patient ID field.
- [Flight delay prediction](#): Uses quantile normalization to sort cases into 10 bins.
- [Twitter sentiment analysis](#): Scores are grouped into five bins representing the ranking scores.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Effects of different normalization methods

If you select the option, **Quantile normalization**, values are transformed before binning. Thus, the method you choose for normalization has a strong effect on the numerical values.

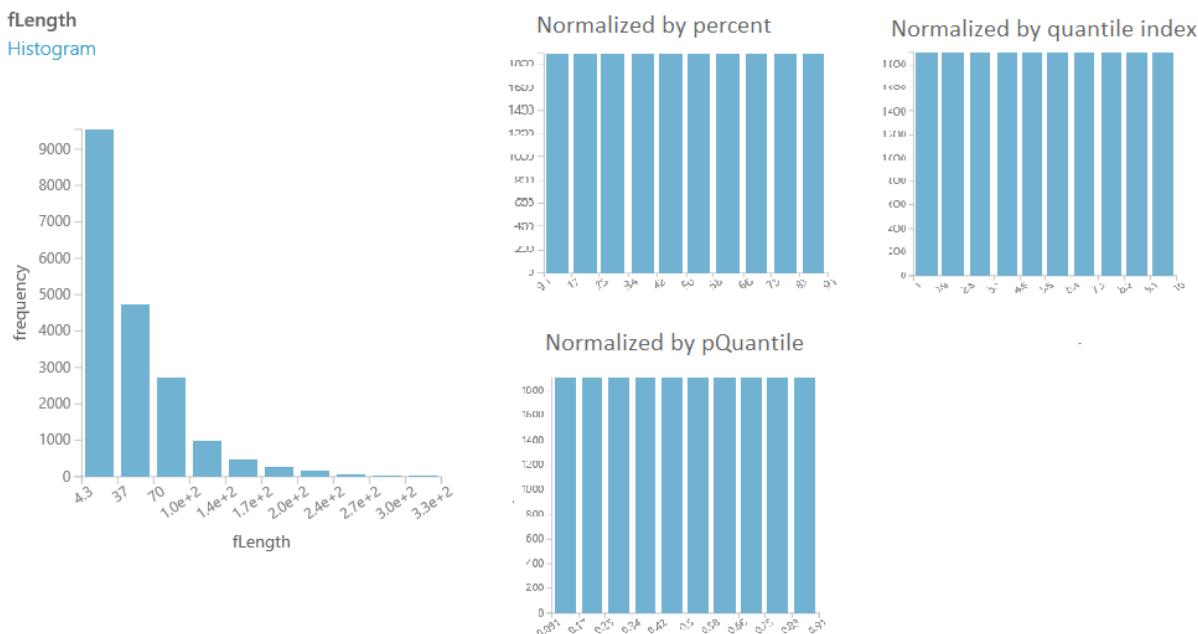
For example, the following table illustrates how the values in a single column, `fLength`, from the [Telescope](#) dataset is transformed with each of the normalization methods. The column, `fLength`, was chosen randomly for illustration of the output values from each option, and does not have a normal distribution.

SOURCE (FLENGTH)	PQUANTILE	QUANTILEINDEX	PERCENT
28.7967	0.363636	4	36.363636

SOURCE (FLENGTH)	PQUANTILE	QUANTILEINDEX	PERCENT
31.6036	0.454545	5	45.454545
162.052	0.909091	10	90.909091
23.8172	0.272727	3	27.272727

The binning results are similar for each method.

The following graphic shows the distribution of values in the column before and after binning, using the default of 10 bins.



Implementation details

- During quantization, each number is mapped to a bin, by comparing its value against the values of the bin edges.

For example, if the value is 1.5 and the bin edges are 1, 2, and 3, the element would be mapped to bin number 2. Value 0.5 would be mapped to bin number 1 (the underflow bin), and value 3.5 would be mapped to bin number 4 (the overflow bin).

- If the column to bin (quantize) is sparse, then the bin index offset (the quantile offset) is used when the resulting column is populated. The offset is chosen so that sparse 0 always goes to the bin with an index of 0 (in other words, the quantile with value 0).
- Sparse zeros are propagated from the input to the output column.
- Processing of dense columns always produces results with a minimum bin index equal to 1; that is, the minimum quantile value equals the minimum value in the column. At the same time, processing of a sparse column produces a result with variable minimum bin index (minimum quantile value).
- All NaNs and missing values are propagated from the input column to the output column. The only exception is the case when the module returns quantile indexes. In this case all NaNs are promoted to missing values.
- Bin indices are 1-based. This is the natural convention for quantiles (1st quantile, 2nd quantile, and so on). The only exception is the case when the column to bin is sparse.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Dataset to be analyzed

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Binning mode	List	QuantizationMode	Quantiles	Choose a binning method
Columns to bin	any	ColumnSelection	NumericAll	Choose columns for quantization
Output mode	any	OutputTo		Indicate how quantized columns should be output
Tag columns as categorical	any	Boolean	true	Indicate whether output columns should be tagged as categorical
Number of bins	>=1	Integer	10	Specify the desired number of bins
Quantile normalization	any	BinningNormalization		Choose the method for normalizing quantiles
First edge position	any	Float	0.0	Specify the value for the first bin edge
Bin width	any	Float	0.5	Specify a custom bin width
Last edge position	any	Float	1.0	Specify the value for the last bin edge
Comma-separated list of bin edges	any	String		Type a comma-separated list of numbers to use as bin edges

Outputs

NAME	TYPE	DESCRIPTION
Quantized dataset	Data Table	Dataset with quantized columns
Binning transformation	ITransform interface	Transformation that applies quantization to the dataset

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.
Error 0011	Exception occurs if passed column set argument does not apply to any of dataset columns.
Error 0021	Exception occurs if number of rows in some of the datasets passed to the module is too small.
Error 0024	Exception occurs if dataset does not contain a label column.
Error 0020	Exception occurs if number of columns in some of the datasets passed to the module is too small.
Error 0038	Exception occurs if number of elements expected should be an exact value, but is not.
Error 0005	Exception occurs if parameter is less than a specific value.
Error 0002	Exception occurs if one or more parameters could not be parsed or converted from specified type into required by target method type.
Error 0019	Exception occurs if column is expected to contain sorted values, but it does not.
Error 0039	Exception occurs if operation has failed.
Error 0075	Exception occurs when an invalid binning function is used when quantizing a dataset.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Scale and Reduce](#)

[Normalize Data](#)

[Clip Values](#)

Normalize Data

3/10/2021 • 6 minutes to read • [Edit Online](#)

Rescales numeric data to constrain dataset values to a standard range

Category: [Data Transformation / Scale and Reduce](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Normalize Data** module in Azure Machine Learning Studio (classic), to transform a dataset through *normalization*.

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Normalization is also required for some algorithms to model the data correctly.

For example, assume your input dataset contains one column with values ranging from 0 to 1, and another column with values ranging from 10,000 to 100,000. The great difference in the *scale* of the numbers could cause problems when you attempt to combine the values as features during modeling.

Normalization avoids these problems by creating new values that maintain the general distribution and ratios in the source data, while keeping values within a scale applied across all numeric columns used in the model.

This module offers several options for transforming numeric data:

- You can change all values to a 0-1 scale, or transform the values by representing them as percentile ranks rather than absolute values.
- You can apply normalization to a single column, or to multiple columns in the same dataset.
- If you need to repeat the experiment, or apply the same normalization steps to other data, you can save the steps as a normalization transform, and apply it to other datasets that have the same schema.

WARNING

Some algorithms require that data be normalized before training a model. Other algorithms perform their own data scaling or normalization. Therefore, when you choose a machine learning algorithm to use in building a predictive model, be sure to review the data requirements of the algorithm before applying normalization to the training data.

How to configure Normalize Data

You can apply only one normalization method at a time using this module. Therefore, the same normalization method is applied to all columns that you select. To use different normalization methods, use a second instance of **Normalize Data**.

1. Add the **Normalize Data** module to your experiment. You can find the module in Azure Machine Learning Studio (classic), under **Data Transformation**, in the **Scale and Reduce** category.
2. Connect a dataset that contains at least one column of all numbers.
3. Use the Column Selector to choose the numeric columns to normalize. If you don't choose individual columns, by default **all** numeric type columns in the input are included, and the same normalization process is applied to all selected columns.

This can lead to strange results if you include numeric columns that shouldn't be normalized! Always check the columns carefully.

If no numeric columns are detected, check the column metadata to verify that the data type of the column is a supported numeric type.

TIP

To ensure that columns of a specific type are provided as input, try using the [Select Columns in Dataset](#) module before **Normalize Data**.

4. **Use 0 for constant columns when checked:** Select this option when any numeric column contains a single unchanging value. This ensures that such columns are not used in normalization operations.
5. From the **Transformation method** dropdown list, choose a single mathematical functions to apply to all selected columns.
 - **Zscore:** Converts all values to a z-score.

The values in the column are transformed using the following formula:

$$z = \frac{x - \text{mean}(x)}{\text{stddev}(x)}$$

Mean and standard deviation are computed for each column separately. Population standard deviation is used.

- **MinMax:** The min-max normalizer linearly rescales every feature to the [0,1] interval.

Rescaling to the [0,1] interval is done by shifting the values of each feature so that the minimal value is 0, and then dividing by the new maximal value (which is the difference between the original maximal and minimal values).

The values in the column are transformed using the following formula:

$$z = \frac{x - \text{min}(x)}{\text{max}(x) - \text{min}(x)}$$

- **Logistic:** The values in the column are transformed using the following formula:

$$z = \frac{1}{1 + \exp(-x)}$$

- **LogNormal:** This option converts all values to a lognormal scale.

The values in the column are transformed using the following formula:

$$z = \text{Lognormal.CDF}(x; \mu, \sigma)$$

Here μ and σ are the parameters of the distribution, computed empirically from the data as maximum likelihood estimates, for each column separately.

- **TanH:** All values are converted to a hyperbolic tangent.

The values in the column are transformed using the following formula:

$$p(k|x;\theta) = \frac{[E(Y|x)]^k e^{-E(Y|x)}}{k!}$$

6. Run the experiment, or double-click the **Normalize Data** module and select **Run Selected**.

Results

The **Normalize Data** module generates two outputs:

- To view the transformed values, right-click the module, select **Transformed dataset**, and click **Visualize**.

By default, values are transformed in place. If you want to compare the transformed values to the original values, use the [Add Columns](#) module to recombine the datasets and view the columns side-by-side.

- To save the transformation so that you can apply the same normalization method to another similar dataset, right-click the module, select **Transformation function**, and click **Save as Transform**.

You can then load the saved transformations from the **Transforms** group of the left navigation pane and apply it to a dataset with the same schema by using [Apply Transformation](#).

Examples

For examples of how normalization is used in machine learning, see the [Azure AI Gallery](#):

- **Credit risk prediction:** In this sample, normalization is applied to all numeric data except the class column, the credit risk score. This example uses the `tanh` transformation, which converts all numeric features to values within a range of 0-1.

Technical notes

This module supports only the standard normalization methods listed in the [How to](#) section, and does not support matrix normalization or other complex transforms.

If you need to create a custom normalization method, you can use the [Execute R Script](#) or [Execute Python Script](#) modules to compute and apply the transformation.

Algorithms that apply normalization

Normalizing features so that they use a common scale is a general requirement for many machine learning algorithms.

- In linear classification algorithms, instances are viewed as vectors in multi-dimensional space. Since the range of values of raw data varies widely, some objective functions do not work properly without normalization. For example, if one of the features has a broad range of values, the distances between points is governed by this particular feature.

Therefore, numeric features should be normalized so that each feature contributes approximately

proportionately to the final distance. This can provide significant speedup and accuracy benefits.

- When using the **Logistic Regression** and **Averaged Perceptron** algorithms, by default, features are normalized before training.

Further reading and resources

If you are unsure which type of normalization suits your data, see these resources:

- [Recommend Modules for My Data](#): This custom module by a member of the Azure ML team evaluates your dataset and recommends steps for cleaning and scaling data.
- [Feature scaling](#): This article in Wikipedia explains the basic methods used for normalizing numeric data.
- [Data Preparation for Data Mining](#) covers many data preparation steps in depth. See Chapter 7 for a discussion of data normalization.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Transformation method	any	TransformationMethods	ZScore	Choose the mathematical method used for scaling
Columns to transform	any	ColumnSelection	NumericAll	Select all columns to which the selected transformation should be applied

Outputs

NAME	TYPE	DESCRIPTION
Transformed dataset	Data Table	Transformed dataset
Transformation function	ITransform interface	Definition of the transformation function, which can be applied to other datasets

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.

EXCEPTION	DESCRIPTION
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.
Error 0020	Exception occurs if number of columns in some of the datasets passed to the module is too small.
Error 0021	Exception occurs if number of rows in some of the datasets passed to the module is too small.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Scale and Reduce](#)

Principal Component Analysis

3/10/2021 • 4 minutes to read • [Edit Online](#)

Computes a set of features with reduced dimensionality for more efficient learning

Category: [Data Transformation / Sample and Split](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Principal Component Analysis** module in Azure Machine Learning Studio (classic) to reduce the dimensionality of your training data. The module analyzes your data and creates a reduced feature set that captures all the information contained in the dataset, but in a smaller number of features.

The module also creates a transformation that you can apply to new data, to achieve a similar reduction in dimensionality and compression of features, without requiring additional training.

More about Principal Component Analysis

Principal Component Analysis (PCA) is a popular technique in machine learning. It relies on the fact that many types of vector-space data are compressible, and that compression can be most efficiently achieved by sampling.

Added benefits of PCA are improved data visualization, and optimization of resource use by the learning algorithm.

The **Principal Component Analysis** module in Azure Machine Learning Studio (classic) takes a set of feature columns in the provided dataset, and creates a projection of the feature space that has lower dimensionality. The algorithm uses randomization techniques to identify a feature subspace that captures most of the information in the complete feature matrix. Hence, the transformed data matrices capture the variance in the original data while reducing the effect of noise and minimizing the risk of overfitting.

For general information about principal component analysis (PCA) see this [Wikipedia article](#). For information about the PCA approaches used in this module, see these articles:

- [Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions](#). Halko, Martinsson, and Tropp, 2010.
- [Combining Structured and Unstructured Randomness in Large Scale PCA](#)Combining Structured and Unstructured Randomness in Large Scale PCA. Karampatziakis and Mineiro, 2013.

How to configure Principal Component Analysis

1. Add the **Principal Component Analysis** module to your experiment. You can find it in under **Data Transformation**, in the **Scale and Reduce** category.
2. Connect the dataset you want to transform, and choose the feature columns to analyze.

If it is not already clear which columns are features and which are labels, we recommend that you use the [Edit Metadata](#) module to mark the columns in advance.

3. **Number of dimensions to reduce to:** Type the desired number of columns in the final output. Each column represents a dimension capturing some part of the information in the input columns.

For example, if the source dataset has eight columns and you type `3`, three new columns are returned that capture the information of the eight selected columns. The columns are named `col1`, `col2`, and `col3`. These columns do not map directly to the source columns; instead, the columns contain an approximation of the feature space described by the original columns 1-8.

TIP

The algorithm functions optimally when the number of reduced dimensions is much smaller than the original dimensions.

4. **Normalize dense dataset to zero mean:** Select this option if the dataset is dense, meaning it contains few missing values. If selected, the module normalizes the values in the columns to a mean of zero before any other processing.

For sparse datasets, this option should not be selected. If a sparse dataset is detected, the parameter is overridden.

5. Run the experiment.

Results

The module outputs a reduced set of columns that you can use in creating a model. You can save the output as a new dataset or use it in your experiment.

Optionally, you can save the analysis process as a saved transform, to apply to another dataset using [Apply Transformation](#).

The dataset you apply the transformation to must have the same schema as the original dataset.

Examples

For examples of how Principal Component Analysis is used in machine learning, see the [Azure AI Gallery](#):

- [Clustering: Find Similar Companies](#): Uses Principal Component Analysis to reduce the number of values from text mining to a manageable number of features.

Although in this sample PCA is applied using a custom R script, it illustrates how PCA is typically used.

Technical notes

There are two stages to computation of the lower-dimensional components.

- The first is to construct a low-dimensional subspace that captures the action of the matrix.
- The second is to restrict the matrix to the subspace and then compute a standard factorization of the reduced matrix.

Expected inputs

NAME	TYPE	DESCRIPTION
------	------	-------------

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Dataset whose dimensions are to be reduced

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DESCRIPTION	DEFAULT
Selected columns	ColumnSelection		Required		Selected columns to apply PCA to
Number of dimensions to reduce to	Integer	>=1	Required		The number of desired dimensions in the reduced dataset
Normalize dense dataset to zero mean	Boolean		Required	true	Indicate whether the input columns will be mean normalized for dense datasets (for sparse data parameter is ignored)

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with reduced dimensions
PCA Transformation	ITransform interface	Transformation which when applied to dataset will give new dataset with reduced dimensions

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Sample and Split](#)
[Feature Selection](#)

Feature Selection modules

3/10/2021 • 7 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that you can use for feature selection.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Feature selection is an important tool in machine learning. Machine Learning Studio (classic) provides multiple methods for performing feature selection. Choose a feature selection method based on the type of data that you have, and the requirements of the statistical technique that's applied.

This article covers:

- [What is feature selection](#)
- [Feature selection modules in Azure Machine Learning](#)
- [How to use feature selection](#)
- [Algorithms that include feature selection](#)

Each feature selection module in Machine Learning Studio (classic) uses a dataset as input. Then, the module applies well-known statistical methods to the data columns that are provided as input. The output is a set of metrics that can help you identify the columns that have the best information value.

About feature selection

In machine learning and statistics, *feature selection* is the process of selecting a subset of relevant, useful features to use in building an analytical model. Feature selection helps narrow the field of data to the most valuable inputs. Narrowing the field of data helps reduce noise and improve training performance.

Often, features are created from raw data through a process of feature engineering. For example, a time stamp in itself might not be useful for modeling until the information is transformed into units of days, months, or categories that are relevant to the problem, such as holiday versus working day.

New users of machine learning might be tempted to include all data that's available. They might expect that the algorithm will find something interesting by using more data. However, feature selection can usually improve your model, and prevent common problems:

- The data contains redundant or irrelevant features, which provide no more information than the currently selected features.
- The data contains irrelevant features that provide no useful information in any context. Including irrelevant fields not only increases the time required to train the data, but also can lead to poor results.
- With some algorithms, having duplicate information in the training data can lead to a phenomenon called *multicollinearity*. In multicollinearity, the presence of two highly correlated variables can cause the calculations for other variables to become much less accurate.

TIP

Some machine learning algorithms in Machine Learning Studio (classic) also use feature selection or dimensionality reduction as part of the training process. When you use these learners, you can skip the feature selection process and let the algorithm decide the best inputs.

Use feature selection in an experiment

Feature selection typically is performed when you are exploring data and developing a new model. Keep these tips in mind when you use feature selection:

- When testing, add feature selection to your experiment to generate scores that inform your decision of which columns to use.
- Remove feature selection from the experiment when you operationalize a model.
- Run feature selection periodically to ensure that the data and best features haven't changed.

Feature selection is different from feature engineering, which focuses on creating new features out of existing data.

Resources

- For a discussion of the different ways that you can engineer features or select the best features as part of the data science process, see [Feature engineering in data science](#).
- For a walkthrough of feature selection in the data science process, see [Filter features from your data - Feature selection](#).

Feature selection methods in Machine Learning Studio (classic)

The following feature selection modules are provided in Machine Learning Studio (classic).

Filter Based Feature Selection

When you use the [Filter Based Feature Selection](#) module, you can choose from among well-known feature selection methods. The module outputs both the feature selection statistics and the filtered dataset.

Your choice of a filter selection method depends in part on what sort of input data you have.

METHOD	SUPPORTED FEATURE INPUTS	SUPPORTED LABELS
Pearson's correlation	Numeric and logical columns only	A single numeric or logical column
Mutual information score	All data types	A single column of any data type
Kendall's correlation coefficient	Numeric and logical columns only	A single numeric or logical column Columns should have values that can be ranked
Spearman's correlation coefficient	Numeric and logical columns only	A single numeric or logical column
Chi-squared statistic	All data types	A single column of any data type
Fisher score	Numeric and logical columns only	A single numeric or logical column String columns are assigned a score of 0

METHOD	SUPPORTED FEATURE INPUTS	SUPPORTED LABELS
Count based feature selection	All data types	A label column is not required

Fisher Linear Discriminant Analysis

Linear Discriminant Analysis is a supervised learning technique that you can use to classify numerical variables in conjunction with a single categorical target. The method is useful for feature selection because it identifies the combination of features or parameters that best separates the groups.

You can use the [Fisher Linear Discriminant Analysis](#) module to generate a set of scores for review, or you can use the replacement dataset that's generated by the module for training.

Permutation Feature Importance

Use the [Permutation Feature Importance](#) module to simulate the effect of any set of features on your dataset. The module computes performance scores for a model based on random shuffling of feature values.

The scores that the module returns represent the potential change in the accuracy of a trained model if values change. You can use the scores to determine the effect of individual variables on the model.

Machine learning algorithms that incorporate feature selection

Some machine learning algorithms in Machine Learning Studio (classic) optimize feature selection during training. They might also provide parameters that help with feature selection. If you're using a method that has its own heuristic for choosing features, it's often better to rely on that heuristic instead of preselecting features.

These algorithms and feature selection methods are used internally:

- **Boosted decision tree models for classification and regression**

In these modules, a feature summary is created internally. Features that have a weight of 0 aren't used by any tree splits. When you visualize the best trained model, you can look at each of the trees. If a feature is never used in any tree, the feature is likely a candidate for removal. To optimize selection, it's also a good idea to use parameter sweeping.

- **Logistic regression models and linear models**

The modules for multiclass and binary logistic regression support L1 and L2 regularization.

Regularization is a way of adding constraints during training to manually specify an aspect of the learned model. Regularization typically is used to avoid overfitting. Machine Learning Studio (classic) supports regularization for the L1 or L2 norms of the weight vector in linear classification algorithms:

- L1 regularization is useful if the goal is to have a model that's as sparse as possible.
- L2 regularization prevents any single coordinate in the weight vector from growing too much in magnitude. It's useful if the goal is to have a model with small overall weights.
- L1-regularized logistic regression is more aggressive about assigning a weight of 0 to features. It's useful in identifying features that can be removed.

Technical notes

All feature selection modules and analytical methods that support numeric and logical columns also support date-time and timespan columns. These columns are treated as simple numeric columns in which each value equals the number of ticks.

Related tasks

The following modules aren't in the **Feature Selection** category, but you can use them for related tasks. The modules can help you reduce the dimensionality of your data or find correlations:

- [Principal Component Analysis](#)

If you have a dataset that has many columns, use the [Principal Component Analysis](#) module to detect the columns that contain the most information about the original data.

This module is in the [Data Transformation](#) category, under [Scale and Reduce](#).

- [Learning with Counts](#)

Count-based featurization is a new technique that you can use to determine useful features by using large datasets. Use these modules to analyze datasets to find the best features, save a set of features to use with new data, or update an existing feature set.

- [Compute Linear Correlation](#)

Use this module to compute a set of Pearson correlation coefficients for each possible pair of variables in the input dataset. The Pearson correlation coefficient, also called Pearson's R test, is a statistical value that measures the linear relationship between two variables.

This module is in the [Statistical Functions](#) category.

List of modules

The **Feature Selection** category includes these modules:

- [Filter Based Feature Selection](#): Identifies the features in a dataset that have the greatest predictive power.
- [Fisher Linear Discriminant Analysis](#): Identifies the linear combination of feature variables that can best group data into separate classes.
- [Permutation Feature Importance](#): Computes the permutation feature importance scores of feature variables for a trained model and test dataset.

See also

- [Module categories and descriptions](#)

Filter Based Feature Selection

3/10/2021 • 12 minutes to read • [Edit Online](#)

Identifies the features in a dataset with the greatest predictive power

Category: [Feature Selection Modules](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Filter Based Feature Selection](#) module in Azure Machine Learning Studio (classic), to identify the columns in your input dataset that have the greatest predictive power.

In general, *feature selection* refers to the process of applying statistical tests to inputs, given a specified output, to determine which columns are more predictive of the output. The [Filter Based Feature Selection](#) module provides multiple feature selection algorithms to choose from, including correlation methods such as Pearson's or Kendall's correlation, mutual information scores, and chi-squared values. Azure Machine Learning also supports feature value counts as an indicator of information value.

When you use the [Filter Based Feature Selection](#) module, you provide a dataset, identify the column that contains the label or dependent variable, and then specify a single method to use in measuring feature importance.

The module outputs a dataset that contains the best feature columns, as ranked by predictive power. It also outputs the names of the features and their scores from the selected metric.

What is filter-based feature selection and why use it?

This module for feature selection is called "filter-based" because you use the selected metric to identify irrelevant attributes, and filter out redundant columns from your model. You choose a single statistical measure that suits your data, and the module calculates a score for each feature column. The columns are returned ranked by their feature scores.

By choosing the right features, you can potentially improve the accuracy and efficiency of classification.

You typically use only the columns with the best scores to build your predictive model. Columns with poor feature selection scores can be left in the dataset and ignored when you build a model.

How to choose a feature selection metric

The [Filter-Based Feature Selection](#) provides a variety of metrics for assessing the information value in each column. This section provides a general description of each metric, and how it is applied. Additional requirements for using each metric are stated in the [Technical Notes](#) section and in the [Instructions](#) for configuring each module.

- **Pearson Correlation**

Pearson's correlation statistic, or Pearson's correlation coefficient, is also known in statistical models as the r value. For any two variables, it returns a value that indicates the strength of the correlation

Pearson's correlation coefficient is computed by taking the covariance of two variables and dividing by the product of their standard deviations. The coefficient is not affected by changes of scale in the two variables.

- **Mutual Information**

The mutual information score measures the contribution of a variable towards reducing uncertainty about the value of another variable: namely, the label. Many variations of the mutual information score have been devised to suit different distributions.

The mutual information score is particularly useful in feature selection because it maximizes the mutual information between the joint distribution and target variables in datasets with many dimensions.

- **Kendall Correlation**

Kendall's rank correlation is one of several statistics that measure the relationship between rankings of different ordinal variables or different rankings of the same variable. In other words, it measures the similarity of orderings when ranked by the quantities. Both this coefficient and Spearman's correlation coefficient are designed for use with non-parametric and non-normally distributed data.

- **Spearman Correlation**

Spearman's coefficient is a nonparametric measure of statistical dependence between two variables, and is sometimes denoted by the Greek letter rho. The Spearman's coefficient expresses the degree to which two variables are monotonically related. It is also called Spearman rank correlation, because it can be used with ordinal variables.

- **Chi Squared**

The two-way chi-squared test is a statistical method that measures how close expected values are to actual results. The method assumes that variables are random and drawn from an adequate sample of independent variables. The resulting chi-squared statistic indicates how far results are from the expected (random) result.

- **Fisher Score**

The Fisher score (also called the Fisher method, or Fisher combined probability score) is sometimes termed the information score, because it represents the amount of information that one variable provides about some unknown parameter on which it depends.

The score is computed by measuring the variance between the expected value of the information and the observed value. When variance is minimized, information is maximized. Since the expectation of the score is zero, the Fisher information is also the variance of the score.

- **Count Based**

Count-based feature selection is a simple yet relatively powerful way of finding information about predictors. The basic idea underlying count-based featurization is simple: by calculating counts of individual values within a column, you can get an idea of the distribution and weight of values, and from this, understand which columns contain the most important information.

Count-based feature selection is a non-supervised method of feature selection, meaning you don't need a label column. This method also reduces the dimensionality of the data without losing information.

For more information about how count-based features are created and why they are useful in machine learning, see [Learning with Counts](#).

TIP

If you need a different option for custom feature selection method, use the [Execute R Script](#) module.

How to configure Filter-Based Feature Selection

This module provides two methods for determining feature scores:

- [Generate feature scores using a traditional statistical metric](#)

You choose a standard statistical metric, and the module computes the correlation between a pair of columns, the label column and a feature column

- [Use count-based feature selection](#)

With the count-based method, the module calculates a score based purely on the values in the column.

Generate feature scores using a traditional statistical metric

1. Add the **Filter-Based Feature Selection** module to your experiment. You can find it in the **Feature Selection** category in Studio (classic).

2. Connect an input dataset that contains at least two columns that are potential features.

To ensure that a column should be analyzed and a feature score generated, use the [Edit Metadata](#) module to set the **IsFeature** attribute.

IMPORTANT

Ensure that the columns you are providing as input are potential features. For example, a column that contains a single value has no information value.

If you know there are columns that would make bad features, you can remove them from the column selection. You could also use the [Edit Metadata](#) module to flag them as **Categorical**.

3. For **Feature scoring method**, choose one of the following established statistical methods to use in calculating scores.

METHOD	REQUIREMENTS
Pearson Correlation	Label can be text or numeric. Features must be numeric.
Mutual Information	Labels and features can be text or numeric. Use this method for computing feature importance for two categorical columns.
Kendall Correlation	Label can be text or numeric but features must be numeric.
Spearman Correlation	Label can be text or numeric but features must be numeric.
Chi Squared	Labels and features can be text or numeric. Use this method for computing feature importance for two categorical columns.

METHOD	REQUIREMENTS
Fisher Score	Label can be text or numeric but features must be numeric.
Counts	See: To use Count-Based Feature Selection

TIP

If you change the selected metric, all other selections will be reset, so be sure to set this option first!)

4. Select the **Operate on feature columns only** option to generate a score only for those columns that have been previously marked as features.

If you deselect this option, the module will create a score for any column that otherwise meets the criteria, up to the number of columns specified in **Number of desired features**.

5. For **Target column**, click **Launch column selector** to choose the label column either by name or by its index (indexes are one-based).

A label column is required for all methods that involve statistical correlation. The module returns a design-time error if you choose no label column or multiple label columns.

6. For **Number of desired features**, type the number of feature columns you want returned as a result.

- The minimum number of features you can specify is 1, but we recommend that you increase this value.
- If the specified number of desired features is greater than the number of columns in the dataset, then all features are returned, even those with zero scores.
- If you specify fewer result columns than there are feature columns, the features are ranked by descending score, and only the top features are returned.

7. Run the experiment, or select the [Filter Based Feature Selection](#) module and then click **Run selected**.

Results of feature selection

After processing is complete:

- To see a complete list of the feature columns that were analyzed, and their scores, right-click the module, select **Features**, and click **Visualize**.
- To view the dataset that is generated based on your feature selection criteria, right-click the module, select **Dataset**, and click **Visualize**.

If the dataset contains fewer columns than you expected, check the module settings, and the data types of the columns provided as input. For example, if you set **Number of desired features** to 1, the output dataset contains just two columns: the label column, and the most highly ranked feature column.

Use count-based feature selection

1. Add the **Filter-Based Feature Selection** module to your experiment. You can find it in the list of modules in Studio (classic), in the **Feature Selection** group.
2. Connect an input dataset that contains at least two columns that are possible features.
3. Select **Count Based** from the list of statistical methods in the **Feature scoring method** dropdown list.
4. For **Minimum number of non-zero elements**, indicate the minimum number of feature columns to

include in the output.

By default, the module outputs all columns that meet the requirements. The module cannot output any column that gets a score of zero.

5. Run the experiment, or select just the module, and click **Run Selected**.

Results of count-based feature selection

- To see the list of feature columns with their scores, right-click the module, select **Features**, and click **Visualize**.
- To see the dataset containing the analyzed columns, right-click the module, select **Dataset**, and click **Visualize**.

Unlike other methods, the **Count Based** feature selection method does not rank the variables by highest scores, but returns all variables with a non-zero score, in their original order.

String features always get a zero (0) score and are thus are not output.

Examples

You can see examples of how feature selection is used in the [Azure AI Gallery](#):

- [Text Classification](#): In the third step of this sample, **Filter-Based Feature Selection** is used to identify the 15 best features. Feature hashing is used to convert the text documents to numeric vectors. Pearson's correlation is then used on the vector features.
- [Machine learning feature selection and feature engineering](#): This article provides an introduction to feature selection and feature engineering in machine learning.

To see examples of feature scores, see [Table of scores compared](#).

Technical notes

You can find this module under **Data Transformation**, in the **Filters** category.

Implementation details

If you use Pearson Correlation, Kendall Correlation, or Spearman Correlation on a numeric feature and a categorical label, the feature score is calculated as follows:

1. For each level in the categorical column, compute the conditional mean of numeric column.
2. Correlate the column of conditional means with the numeric column.

Requirements

- A feature selection score cannot be generated for any column that is designated as a **label** or as a **score** column.
- If you attempt to use a scoring method with a column of a data type not supported by the method, either the module will raise an error, or a zero score will be assigned to the column.
- If a column contains logical (true/false) values, they are processed as True = 1 and False = 0.
- A column cannot be a feature if it has been designated as a **Label** or a **Score**.

How missing values are handled

- You cannot specify as a target (label) column any column that has all missing values.
- If a column contains missing values, they are ignored when computing the score for the column.

- If a column designated as a feature column has all missing values, a zero score is assigned.

Table of scores compared

To give you an idea of how the scores compare when using different metrics, the following table presents some feature selection scores from multiple features in the automobile price dataset, given the dependent variable **highway-mpg**.

FEATURE COLUMN	PEARSON SCORE	COUNT SCORE	KENDALL SCORE	MUTUAL INFORMATION
highway-mpg	1	205	1	1
city-mpg	0.971337	205	0.892472	0.640386
curb-weight	0.797465	171	0.673447	0.326247
horsepower	0.770908	203	0.728289	0.448222
price	0.704692	201	0.651805	0.321788
length	0.704662205	205	0.53193	0.281317
engine-size	0.67747	205	0.581816	0.342399
width	0.677218	205	0.525585	0.285006
bore	0.594572	201	0.467345	0.263846
wheel-base	0.544082	205	0.407696	0.250641
compression-ratio	0.265201	205	0.337031	0.288459
fuel-system	na	na	na	0.308135
make	na	na	na	0.213872
drive-wheels	na	na	na	0.213171
height	na	na	na	0.1924
normalized-losses	na	na	na	0.181734
symboling	na	na	na	0.159521
num-of-cylinders	na	na	na	0.154731
engine-type	na	na	na	0.135641
aspiration	na	na	na	0.068217
body-style	na	na	na	0.06369
fuel-type	na	na	na	0.049971

FEATURE COLUMN	PEARSON SCORE	COUNT SCORE	KENDALL SCORE	MUTUAL INFORMATION
num-of-doors	na	na	na	0.017459
engine-location	na	na	na	0.010166

- Mutual information scores can be created for all column types, including strings.
- The other scores included in this table, such as Pearson's correlation or count-based feature selection, require numeric values. String features get a score of 0 and hence are not included in the output. For exceptions, see the [Technical Notes](#) section.
- The count-based method does not treat a label column any differently from feature columns.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Feature scoring method	List	Scoring method		Choose the method to use for scoring
Operate on feature columns only	Any	Boolean	true	Indicate whether to use only feature columns in the scoring process
Target column	Any	ColumnSelection	None	Specify the target column
Number of desired features	>=1	Integer	1	Specify the number of features to output in results
Minimum number of non-zero elements	>=1	Integer	1	Specify the number of features to output (for CountBased method)

Outputs

NAME	TYPE	DESCRIPTION
Filtered dataset	Data Table	Filtered dataset
Features	Data Table	Names of output columns and feature selection scores

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Feature Selection](#)

[Fisher Linear Discriminant Analysis](#)

[A-Z Module List](#)

Fisher Linear Discriminant Analysis

3/10/2021 • 5 minutes to read • [Edit Online](#)

Identifies the linear combination of feature variables that can best group data into separate classes

Category: [Feature Selection Modules](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Fisher Linear Discriminant Analysis** module in Azure Machine Learning Studio (classic), to create a new feature dataset that captures the combination of features that best separates two or more classes.

This method is often used for dimensionality reduction, because it projects a set of features onto a smaller feature space while preserving the information that discriminates between classes. This not only reduces computational costs for a given classification task, but can help prevent overfitting.

To generate the scores, you provide a label column and set of numerical feature columns as inputs. The algorithm determines the optimal combination of the input columns that linearly separates each group of data while minimizing the distances within each group. The module returns a dataset containing the compact, transformed features, along with a transformation that you can save and apply to another dataset.

More about linear discriminant analysis

Linear discriminant analysis is similar to analysis of variance (ANOVA) in that it works by comparing the means of the variables. Like ANOVA, it relies on these assumptions:

- Predictors are independent
- The conditional probability density functions of each sample are normally distributed
- Variances among groups are similar

Linear Discriminant Analysis is sometimes abbreviated to LDA, but this is easily confused with *Latent Dirichlet Allocation*. The techniques are completely different, so in this documentation, we use the full names wherever possible.

How to configure Linear Discriminant Analysis

1. Add your input dataset and check that the input data meets these requirements:

- Your data should be as complete as possible. Rows with any missing values are ignored.
- Values are expected to have a normal distribution. Before using **Fisher Linear Discriminant Analysis**, review the data for outliers, or test the distribution.
- You should have fewer predictors than there are samples.
- Remove any non-numeric columns. The algorithm examines all valid numeric columns included in the inputs, and return an error if invalid columns are included. If you need to exclude any numeric

columns, add a [Select Columns in Dataset](#) module before **Fisher Linear Discriminant Analysis**, to create a view that contains only the columns you wish to analyze. You can rejoin the columns later using [Add Columns](#). The original order of rows is preserved.

2. Connect the input data to the **Fisher Linear Discriminant Analysis** module.
3. For **Class labels column**, click **Launch column selector** and choose one label column.
4. For **Number of feature extractors**, type the number of columns that you want as a result.

For example, if your dataset contains eight numeric feature columns, you might type to collapse them into a new, reduced feature space of only three columns.

It is important to understand that the output columns do not correspond exactly to the input columns, but rather represent a compact transformation of the values in the input columns.

If you use 0 as the value for **Number of feature extractors**, and n columns are used as input, n feature extractors are returned, containing new values representing the n -dimensional feature space.

5. Run the experiment.

Results

The algorithm determines the combination of values in the input columns that linearly separates each group of data while minimizing the distances within each group, and creates two outputs:

- **Transformed features**. A dataset containing the specified number of feature extractor columns, named `col1`, `col2`, `col3`, and so forth. The output also includes the class or label variable as well.

You can use this compact set of values for training a model.

- **Fisher linear discriminant analysis transformation**. A transformation that you can save and then apply to a dataset that has the same schema. This is useful if you are analyzing many datasets of the same type and want to apply the same feature reduction to each. The dataset that you apply it to should have the same schema.

Examples

For examples of feature selection in machine learning, see the [Azure AI Gallery](#):

- [Fisher Linear Discriminant Analysis](#): Demonstrates how to use this module for dimensionality reduction.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Usage tips

- This method works only on continuous variables, not categorical or ordinal variables.
- Rows with missing values are ignored when computing the transformation matrix.
- If you save a transformation from an experiment, the transformations computed from the original experiment are reapplied to each new set of data, and are not recomputed. Therefore, if you want to compute a new feature set for each set of data, use a new instance of **Fisher Linear Discriminant Analysis** for each dataset.

Implementation details

The dataset of features is transformed using *eigenvectors*. The eigenvectors for the input dataset are computed based on the provided feature columns, also called a *discrimination matrix*.

The transformation output by the module contains these eigenvectors, which can be applied to transform

another dataset that has the same schema.

For more information about how the eigenvalues are calculated, see this paper (PDF): [Eigenvector-based Feature Extraction for Classification](#). Tymbal, Puuronen et al.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Class labels column	ColumnSelection		Required	None	Select the column that contains the categorical class labels
Number of feature extractors	Integer	>=0	Required	0	Number of feature extractors to use. If zero, then all feature extractors will be used

Outputs

NAME	TYPE	DESCRIPTION
Transformed features	Data Table	Fisher linear discriminant analysis features transformed to eigenvector space
Fisher linear discriminant analysis transformation	ITransform interface	Transformation of Fisher linear discriminant analysis

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Feature Selection](#)

[Filter Based Feature Selection](#)

[Principal Component Analysis](#)

Permutation Feature Importance

3/10/2021 • 3 minutes to read • [Edit Online](#)

Computes the permutation feature importance scores of feature variables given a trained model and a test dataset

Category: [Feature Selection Modules](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Permutation Feature Importance](#) module in Azure Machine Learning Studio (classic), to compute a set of feature importance scores for your dataset. You use these scores to help you determine the best features to use in a model.

In this module, feature values are randomly shuffled, one column at a time, and the performance of the model is measured before and after. You can choose one of the standard metrics provided to measure performance.

The scores that the module returns represent the **change** in the performance of a trained model, after permutation. Important features are usually more sensitive to the shuffling process, and will thus result in higher importance scores.

This article provides a good general overview of permutation feature importance, its theoretical basis, and its applications in machine learning: [Permutation feature importance](#)

How to use Permutation Feature Importance

To generate a set of feature scores requires that you have an already trained model, as well as a test dataset.

1. Add the **Permutation Feature Importance** module to your experiment. You can find this module in the **Feature Selection** category.
2. Connect a trained model to the left input. The model must be a regression model or classification model.
3. On the right input, connect a dataset, preferably one that is different from the dataset used for training the model. This dataset is used for scoring based on the trained model, and for evaluating the model after feature values have been changed.
4. For **Random seed**, type a value to use as seed for randomization. If you specify 0 (the default), a number is generated based on the system clock.

A seed value is optional, but you should provide a value if you want reproducibility across runs of the same experiment.

5. For **Metric for measuring performance**, select a single metric to use when computing model quality after permutation.

Azure Machine Learning Studio (classic) supports the following metrics, depending on whether you are

evaluating a classification or regression model:

- **Classification**

Accuracy, Precision, Recall, Average Log Loss

- **Regression**

Precision, Recall, Mean Absolute Error , Root Mean Squared Error, Relative Absolute Error, Relative Squared Error, Coefficient of Determination

For a more detailed description of these evaluation metrics, and how they are calculated, see [Evaluate](#).

6. Run the experiment.

7. The module outputs a list of feature columns and the scores associated with them, ranked in order of the scores, descending.

Examples

See these sample experiments in the [Azure AI Gallery](#):

- [Permutation Feature Importance](#): Demonstrates how to use this module to rank feature variables of a dataset in order of permutation importance scores.
- [Using the Permutation Feature Importance module](#): Illustrates the usage of this module in a web service.

Technical notes

This section provides implementation details, tips, and answers to frequently asked questions.

How does this compare to other feature selection methods?

Permutation feature importance works by randomly changing the values of each feature column, one column at a time, and then evaluating the model.

The rankings provided by permutation feature importance are often different from the ones you get from [Filter Based Feature Selection](#), which calculates scores **before** a model is created.

This is because permutation feature importance doesn't measure the association between a feature and a target value, but instead captures how much influence each feature has on predictions from the model.

Expected inputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	A trained classification or regression model
Test data	Data Table	Test dataset for scoring and evaluating a model after permutation of feature values

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
------	------	-------	----------	---------	-------------

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Random seed	Integer	>=0	Required	0	Random number generator seed value
Metric for measuring performance	EvaluationMetric Type	select from list	Required	Classification - Accuracy	Select the metric to use when evaluating the variability of the model after permutations

Outputs

NAME	TYPE	DESCRIPTION
Feature importance	Data Table	A dataset containing the feature importance results, based on the selected metric

Exceptions

EXCEPTION	DESCRIPTION
Error 0062	Exception occurs when attempting to compare two models with different learner types.
Error 0024	Exception occurs if dataset does not contain a label column.
Error 0105	Thrown when a module definition file defines an unsupported parameter type
Error 0021	Exception occurs if number of rows in some of the datasets passed to the module is too small.

See also

[Feature Selection](#)

[Filter Based Feature Selection](#)

[Principal Component Analysis](#)

Machine learning modules in Azure Machine Learning Studio (classic)

3/10/2021 • 3 minutes to read • [Edit Online](#)

The typical workflow for machine learning includes many phases:

- Identifying a problem to solve and a metric for measuring results.
- Finding, cleaning, and preparing appropriate data.
- Identifying the best features and engineering new features.
- Building, evaluating, and tuning models.
- Using models to generate predictions, recommendations, and other results.

The modules in this section provide tools for the final phases of machine learning, in which you apply an algorithm to data to train a model. In these final phases, you also generate scores, and then evaluate the accuracy and usefulness of the model.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

List of machine learning tasks by category

- [Initialize Model](#)

Choose from a variety of customizable machine learning algorithms, including [clustering](#), [regression](#), [classification](#), and [anomaly detection](#) models.

- [Train](#)

Provide your data to the configured model to learn from patterns and create statistics that can be used for predictions.

- [Score](#)

Create predictions using the trained models.

- [Evaluate](#)

Measure the accuracy of a trained model, or compare multiple models.

For a detailed description of this experimental workflow, see the [credit risk solution walkthrough](#).

Prerequisites

Before you can get to the fun part of building a model, typically a lot of preparation is required. This section provides links to tools in Machine Learning Studio (classic) that can help you clean up your data, improve the quality of input, and prevent run-time errors.

Data exploration and data quality

Ensure that your data is the right kind of data, the right quantity, and the right quality for the algorithm you've chosen. Understand how much data you have, and how it is distributed. Are there outliers? How were those generated, and what do they mean? Are there any duplicate records?

Handle missing values

Missing values can affect your results in many ways. For example, almost all statistical methods discard cases with missing values. By default, Machine Learning follows these rules when it encounters rows with missing values:

- If data used to train a model has missing values, any rows with missing values are skipped.
- If data used as input when scoring against a model has missing values, the missing values are used as inputs, but nulls are propagated. This usually means that a null is inserted in the results instead of a valid prediction.

Be sure to check your data before training your model. To impute the missing values or correct your data, use this module:

- [Clean Missing Data](#)

Select features and reduce dimensionality

Machine Learning Studio (classic) can help you sift through your data to find the most useful attributes.

- Use tools such as [Fisher Linear Discriminant Analysis](#) or [Filter Based Feature Selection](#) to determine which columns of data have the most predictive power. These tools can also identify columns that should be removed because of data leakage.
- Create or engineer features from existing data. [Normalize data](#) or [group data into bins](#) to make new groupings of data, or standardize the range of numeric values prior to analysis.
- Reduce dimensionality by [grouping categorical values](#), by using [principal component analysis](#), or by [sampling](#).

Choose an appropriate algorithm

The problem you are trying to solve determines both the choice of data to use in analysis, and the choice of an algorithm.

For more information, see [How to choose an algorithm in Azure Machine Learning](#).

Examples

For examples of machine learning in action, see the [Azure AI Gallery](#).

For tips, and a walkthrough of some typical data preparation tasks, see [Walkthroughs executing the Team Data Science Process](#).

See also

- [Data Transformation](#)
- [Data Input and Output](#)
- [Statistical Functions](#)

Machine Learning - Evaluate

3/10/2021 • 5 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that you can use to evaluate a machine learning model. *Model evaluation* is performed after training is complete, to measure the accuracy of the predictions and assess model fit.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

This article also describes the overall process in Machine Learning Studio (classic) for model creation, training, evaluation, and scoring.

Create and use machine learning models in Machine Learning Studio (classic)

The typical workflow for machine learning includes these phases:

1. Choose a suitable algorithm and set initial options.
2. Train the model by using compatible data.
3. Create predictions by using new data that's based on the patterns in the model.
4. Evaluate the model to determine whether the predictions are accurate, the amount of error, and whether overfitting occurs.

Machine Learning Studio (classic) supports a flexible, customizable framework for machine learning. Each task in this process is performed by a specific type of module. The module can be modified, added, or removed without breaking the rest of your experiment.

Use the modules in this category to evaluate an existing model. Model evaluation typically requires some kind of result dataset. If you don't have an evaluation dataset, you can generate results by scoring. You can also use a test dataset, or some other set of data that contains "ground truth" or known expected results.

More about model evaluation

In general, when evaluating a model, your options depend on the type of model you are evaluating, and the metric that you want to use. These topics list some of the most frequently used metrics:

- [Evaluate Model](#)
- [Cross-Validate Model](#)

Machine Learning Studio (classic) also provides a variety of visualizations, depending on the type of model you're using, and how many classes your model is predicting. For help finding these visualizations, see [View evaluation metrics](#).

Interpreting these statistics often requires a greater understanding of the particular algorithm on which the model was trained. For a good explanation of how to evaluate a model, and how to interpret the values that are returned for each measure, see [How to evaluate model performance in Azure Machine Learning](#).

List of modules

The **Machine Learning - Evaluate** category includes the following modules:

- [Cross-Validate Model](#): Cross-validates parameter estimates for classification or regression models by partitioning the data.

Use the [Cross-Validate Model](#) module if you want to test the validity of your training set and the model. Cross-validation partitions the data into folds, and then tests multiple models on combinations of folds.

- [Evaluate Model](#): Evaluates a scored classification or regression model by using standard metrics.

In most cases, you'll use the generic [Evaluate Model](#) module. This is especially true if your model is based on one of the supported classification or regression algorithms.

- [Evaluate Recommender](#): Evaluates the accuracy of recommender model predictions.

For recommendation models, use the [Evaluate Recommender](#) module.

Related tasks

- For clustering models, use the [Assign Data to Clusters](#) module. Then, use the visualizations in that module to see evaluation results.
- You can create custom evaluation metrics. To create custom evaluation metrics, provide R code in the [Execute R Script](#) module, or Python code in the [Execute Python Script](#) module. This option is handy if you want to use metrics that were published as part of open-source libraries, or if you want to design your own metric for measuring model accuracy.

Examples

Interpreting the results of machine learning model evaluation is an art. It requires understanding the mathematical results, in addition to the data and the business problems. We recommend that you review these articles for an explanation of how to interpret results in different scenarios:

- [Choose parameters to optimize your algorithms in Azure Machine Learning](#)
- [Interpret model results in Azure Machine Learning](#)
- [Evaluate model performance in Azure Machine Learning](#)

Technical notes

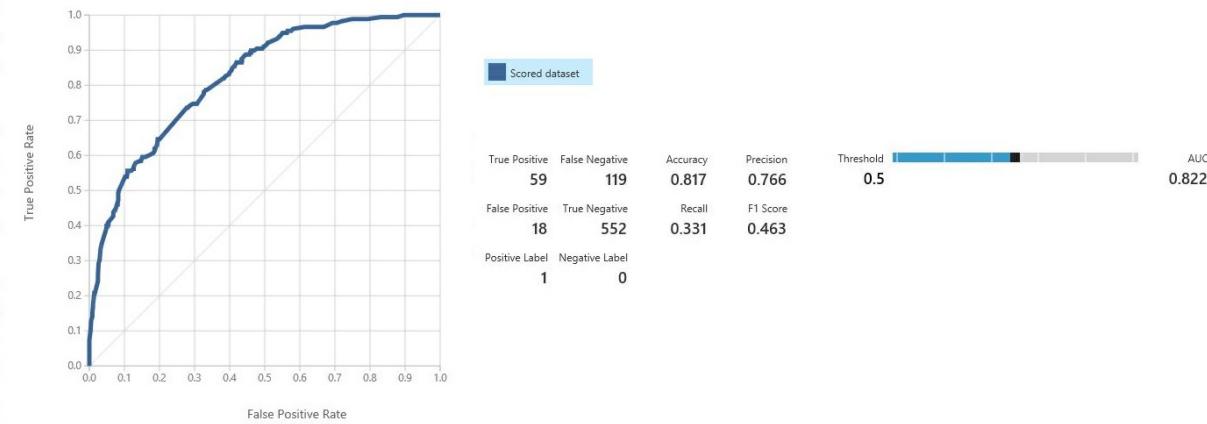
This section contains implementation details, tips, and answers to frequently asked questions.

View evaluation metrics

Learn where to look in Machine Learning Studio (classic) to find the metric charts for each model type.

Two-class classification models

The default view for binary classification models includes an interactive ROC chart and a table of values for the principal metrics.



You have two options for viewing binary classification models:

- Right-click the module output, and then select **Visualize**.
- Right-click the module, select **Evaluation results**, and then select **Visualize**.

You can also use the slider to change the probability **Threshold** value. The threshold determines whether a result should be accepted as true or not. Then, you can see how these values change.

Multiclass classification models

The default metrics view for multi-class classification models includes a confusion matrix for all classes and a set of metrics for the model as a whole.

You have two options for viewing multi-class classification models:

- Right-click the module output, and then select **Visualize**.
- Right-click the module, select **Evaluation results**, and then select **Visualize**.

For simplicity, here are the two results, shown side by side:



Regression models

The metrics view for regression models varies depending on the type of model that you created. The metrics view is based on the underlying algorithm interfaces, and on the best fit for the model metrics.

You have two options for viewing regression models:

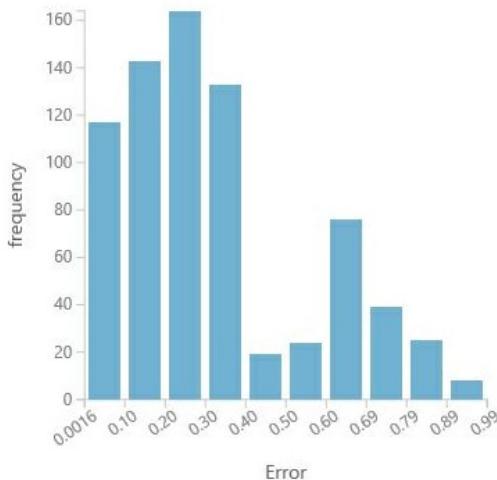
- To view the accuracy metrics in a table, right-click the **Evaluate Model** module's output, and then select

Visualize.

- To view an error histogram with the values, right-click the module, select **Evaluation results**, and then select **Visualize**.

Evaluate - Regression > Evaluate Model > Evaluation results

Error Histogram



Metrics

Mean Absolute Error	0.324342
Root Mean Squared Error	0.397412
Relative Absolute Error	0.894298
Relative Squared Error	0.870944
Coefficient of Determination	0.129056

The **Error Histogram** view can help you understand how error is distributed. It's provided for the following model types, and includes a table of default metrics, such as root mean squared error (RMSE).

- [Boosted Decision Tree Regression](#)
- [Linear Regression](#)
- [Neural Network Regression](#)

The following regression models generate a table of default metrics, along with some custom metrics:

- [Bayesian Linear Regression](#)
- [Decision Forest Regression](#)
- [Fast Forest Quantile Regression](#)
- [Ordinal Regression](#)

Tips for working with the data

To extract the numbers without copying and pasting from the Machine Learning Studio (classic) UI, you can use the new [PowerShell library for Azure Machine Learning](#). You can get metadata and other information for an entire experiment, or from individual modules.

To extract values from an **Evaluate Model** module, you must add a unique comment to the module, for easier identification. Then, use the `Download-AmlExperimentNodeOutput` cmdlet to get the metrics and their values from the visualization in JSON format.

For more information, see [Create machine learning models by using PowerShell](#).

See also

- [Train](#)
- [Score](#)
- [Evaluate](#)
- [Module categories and descriptions](#)
- [A-Z module list](#)

Cross-Validate Model

3/10/2021 • 9 minutes to read • [Edit Online](#)

Cross-validates parameter estimates for classification or regression models by partitioning the data

Category: [Machine Learning / Evaluate](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Cross-Validate Model** module in Azure Machine Learning Studio (classic). *Cross-validation* is an important technique often used in machine learning to assess both the variability of a dataset and the reliability of any model trained using that data.

The **Cross-Validate Model** module takes as input a labeled dataset, together with an untrained classification or regression model. It divides the dataset into some number of subsets (*folds*), builds a model on each fold, and then returns a set of accuracy statistics for each fold. By comparing the accuracy statistics for all the folds, you can interpret the quality of the data set and understand whether the model is susceptible to variations in the data.

Cross-validate also returns predicted results and probabilities for the dataset, so that you can assess the reliability of the predictions.

How cross-validation works

1. Cross validation randomly divides the training data into a number of partitions, also called *folds*.
 - The algorithm defaults to 10 folds if you have not previously partitioned the dataset.
 - To divide the dataset into a different number of folds, you can use the [Partition and Sample](#) module and indicate how many folds to use.
2. The module sets aside the data in fold 1 to use for validation (this is sometimes called the *holdout fold*), and uses the remaining folds to train a model.

For example, if you create five folds, the module would generate five models during cross-validation, each model trained using 4/5 of the data, and tested on the remaining 1/5.
3. During testing of the model for each fold, multiple accuracy statistics are evaluated. Which statistics are used depends on the type of model that you are evaluating. Different statistics are used to evaluate classification models vs. regression models.
4. When the building and evaluation process is complete for all folds, **Cross-Validate Model** generates a set of performance metrics and scored results for all the data. You should review these metrics to see whether any single fold has particularly high or low accuracy

Advantages of cross-validation

A different, and very common way of evaluating a model is to divide the data into a training and test set using [Split Data](#), and then validate the model on the training data. However, cross-validation offers some advantages:

- Cross-validation uses more test data.

Cross-validation measures the performance of the model with the specified parameters in a bigger data space. That is, cross-validation uses the entire training dataset for both training and evaluation, instead of some portion. In contrast, if you validate a model by using data generated from a random split, typically you evaluate the model only on 30% or less of the available data.

However, because cross-validation trains and validates the model multiple times over a larger dataset, it is much more computationally intensive and takes much longer than validating on a random split.

- Cross-validation evaluates the dataset as well as the model.

Cross-validation does not simply measure the accuracy of a model, but also gives you some idea of how representative the dataset is and how sensitive the model might be to variations in the data.

How to use Cross-Validate Model

There are two main ways to use cross-validation.

- [For simple evaluation](#)
- [In combination with a parameter sweep](#)

Cross-validation can take a long time to run if you use a lot of data. Therefore, you might use **Cross-Validate Model** in the initial phase of building and testing your model, to evaluate the goodness of the model parameters (assuming that computation time is tolerable), and then train and evaluate your model using the established parameters with the [Train Model](#) and [Evaluate Model](#) modules.

Simple cross-validation

In this scenario, you both train and test the model using **Cross Validate Model**.

1. Add the **Cross Validate Model** module to your experiment. You can find it in Azure Machine Learning Studio (classic), in the **Machine Learning** category, under **Evaluate**.
2. Connect the output of any [classification](#) or [regression](#) model.

For example, if you are using a **Two Class Bayes Point Machine** for classification, configure the model with the parameters you want, and then drag a connector from the **Untrained model** port of the classifier to the matching port of **Cross Validate Model**.

TIP

The model need not be trained because **Cross-Validate Model** automatically trains the model as part of evaluation.

3. On the **Dataset** port of **Cross Validate Model**, connect any labeled training dataset.
4. In the **Properties** pane of **Cross Validate Model**, click **Launch column selector** and choose the single column that contains the class label, or the predictable value.
5. Set a value for the **Random seed** parameter if you want to be able to repeat the results of cross-validation across successive runs on the same data.
6. Run the experiment.
7. See the [Results](#) section for a description of the reports.

To get a copy of the model for re-use later, right click the output of the module that contains the algorithm (for example, the **Two Class Bayes Point Machine**), and click **Save as Trained Model**.

Cross-validation with a parameter sweep

In this scenario, you use [Tune Model Hyperparameters](#) to identify the best model by conducting a parameter sweep, and then use [Cross Validate Model](#) to check its reliability. This is the easiest way to have Azure Machine Learning identify the best model and then generate metrics for it.

1. Add the dataset for model training, and add one of the machine learning modules that creates a classification or regression model.
2. Add the [Tune Model Hyperparameters](#) module to your experiment. You can find it in the **Machine Learning** category, under **Train**.
3. Attach the classification or regression model to the **Untrained model** input of [Tune Model Hyperparameters](#).
4. Add the [Cross Validate Model](#) module to your experiment. You can find it in Azure Machine Learning Studio (classic), in the **Machine Learning** category, under **Evaluate**.
5. Locate the **Trained best model** output of [Tune Model Hyperparameters](#), and connect it to the **Untrained model** input of [Cross Validate Model](#).
6. Connect the training data to the **Training dataset** input of [Cross Validate Model](#).
7. Run the experiment.
8. After reviewing the results, and the evaluation scores, to get a copy of the best model for later re-use, just right-click the [Tune Model Hyperparameters](#) module, select **Trained best model**, and then click **Save as Trained Model**.

NOTE

You might get different results if you use the input on the [Tune Model Hyperparameters](#) module for **Optional validation dataset**.

That is because when you use this option, you are in effect specifying a static training dataset and testing dataset. Hence, the cross-validation process also uses the specified training and testing datasets, rather than splitting the data into n groups for training and testing. However, metrics are generated on an n -fold basis.

Results

After all iterations are complete, [Cross-Validate Model](#) creates scores for the entire dataset, as well as performance metrics you can use to assess the quality of the model.

Scored results

The first output of the module provides the source data for each row, together with some predicted values and related probabilities.

To view these results, in the experiment, right-click the [Cross-Validate Model](#) module, select **Scored results**, and click **Visualize**.

NEW COLUMN NAME	DESCRIPTION
Fold Assignments	Indicates the 0-based index of the fold each row of data was assigned to during cross-validation.

NEW COLUMN NAME	DESCRIPTION
Scored Labels	This column is added at the end of the dataset, and contains the predicted value for each row
Scored Probabilities	This column is added at the end of the dataset, and indicates the estimated probability of the value in Scored Labels .

Evaluation results

The second report is grouped by folds. Remember that, during execution, **Cross-Validate Model** randomly splits the training data into n folds (by default, 10). In each iteration over the dataset, **Cross-Validate Model** uses one fold as a validation dataset, and uses the remaining $n-1$ folds to train a model. Each of the n models is tested against the data in all the other folds.

In this report, the folds are listed by index value, in ascending order. To order on any other column you can save the results as a dataset.

To view these results, in the experiment, right-click the **Cross-Validate Model** module, select **Evaluation results by fold**, and click **Visualize**.

COLUMN NAME	DESCRIPTION
Fold number	An identifier for each fold. If you created 5 folds, there would be 5 subsets of data, numbered 0 to 4.
Number of examples in fold	The number of rows assigned to each fold. They should be roughly equal.
Model	The algorithm used in the model, identified by the API name

Additionally, the following metrics are included for each fold, depending on the type of model that you are evaluating.

- **Classification models:** Precision, recall, F-score, AUC, average log loss, training log loss
- **Regression models:** Negative log likelihood, mean absolute error, root mean squared error, relative absolute error, and coefficient of determination

Examples

For examples of how cross-validation is used in machine learning, see the [Azure AI Gallery](#):

- [Cross validation for binary classifier](#): Demonstrates how to use cross-validation with a binary classification model.
- [Cross Validation Regression: Auto Imports Dataset](#): Demonstrates how to use cross validation with regression models and how to interpret the results.

Technical notes

- It is a best practice to normalize datasets before using them for cross-validation.
- Because **Cross-Validate Model** trains and validates the model multiple times, it is much more computationally intensive and takes longer to complete than if you validated the model using a randomly divided dataset.
- We recommend that you use **Cross-Validate Model** to establish the goodness of the model given the

specified parameters. Use [Tune Model Hyperparameters](#) to identify the optimal parameters.

- There is no need to split the dataset into training and testing sets when you use cross validation to measure the accuracy of the model.

However, if a validation dataset is provided upstream, the module uses the specified training and testing datasets instead of splitting into n folds. That is, the first dataset is used to train the model for each parameter combination, and the models are evaluated on the validation dataset. See the section on [using a parameter sweep with cross-validation](#).

- Although this article uses older versions of the modules, it has a good explanation of the cross-validation process: [How to choose parameters to optimize your algorithms in Azure Machine Learning](#)

Expected inputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	Untrained model for cross validation on dataset
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Label column	any	ColumnSelection		Select the column that contains the label to use for validation
Random seed	any	Integer	0	Seed value for random number generator This value is optional. If not specified

Outputs

NAME	TYPE	DESCRIPTION
Scored results	Data Table	Results of scoring
Evaluation results by fold	Data Table	Results of evaluation (by fold and entire)

Exceptions

EXCEPTION	DESCRIPTION
Error 0035	Exception occurs if no features were provided for a given user or item.

EXCEPTION	DESCRIPTION
Error 0032	Exception occurs if argument is not a number.
Error 0033	Exception occurs if argument is Infinity.
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0006	Exception occurs if parameter is greater than or equal to the specified value.
Error 0008	Exception occurs if parameter is not in range.
Error 0013	Exception occurs if the learner that is passed to the module has an invalid type.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Evaluate](#)

[Evaluate Recommender](#)

[A-Z Module List](#)

Evaluate Model

3/10/2021 • 9 minutes to read • [Edit Online](#)

Evaluates the results of a classification or regression model with standard metrics

Category: [Machine Learning / Evaluate](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Evaluate Model** module in Azure Machine Learning Studio (classic) to measure the accuracy of a trained model. You provide a dataset containing scores generated from a model, and the **Evaluate Model** module computes a set of industry-standard evaluation metrics.

The metrics returned by **Evaluate Model** depend on the type of model that you are evaluating:

- [Classification Models](#)
- [Regression Models](#)
- [Clustering Models](#)

For recommendation models, use the [Evaluate Recommender](#) module.

TIP

If you are new to model evaluation, we recommend these samples in the Azure AI Gallery, which build a model and then explain how to use the related metrics:

- [Compare regression models](#)
- [Compare binary classifiers](#)
- [Compare multiclass classifiers](#)

We also recommend the video series by Dr. Stephen Elston, as part of the [machine learning course](#) from EdX.

How to use Evaluate Model

There are three ways to use the **Evaluate Model** module:

- Generate scores over your training data, and evaluate the model based on these scores
- Generate scores on the model, but compare those scores to scores on a reserved testing set
- Compare scores for two different but related models, using the same set of data

Use the training data

To evaluate a model, you must connect a dataset that contains a set of input columns and scores. If no other data is available, you can use your original dataset.

1. Connect the **Scored dataset** output of the [Score Model](#) to the input of **Evaluate Model**.

2. Click **Evaluate Model** module, and select **Run selected** to generate the evaluation scores.

Use testing data

A common scenario in machine learning is to separate your original data set into training and testing datasets, using the [Split](#) module, or the [Partition and Sample](#) module.

1. Connect the **Scored dataset** output of the [Score Model](#) to the input of **Evaluate Model**.
2. Connect the output of the Split Data module that contains the testing data to the right-hand input of **Evaluate Model**.
3. Click **Evaluate Model** module, and select **Run selected** to generate the evaluation scores.

Compare scores from two models

You can also connect a second set of scores to **Evaluate Model**. The scores might be a shared evaluation set that has known results, or a set of results from a different model for the same data.

This feature is useful because you can easily compare results from two different models on the same data. Or, you might compare scores from two different runs over the same data with different parameters.

1. Connect the **Scored dataset** output of the [Score Model](#) to the input of **Evaluate Model**.
2. Connect the output of the Score Model module for the second model to the right-hand input of **Evaluate Model**.
3. Right-click **Evaluate Model**, and select **Run selected** to generate the evaluation scores.

Results

After you run **Evaluate Model**, right-click the module and select **Evaluation results** to see the results. You can:

- Save the results as a dataset, for easier analysis with other tools
- Generate a visualization in the Studio (classic) interface

If you connect datasets to both inputs of **Evaluate Model**, the results will contain metrics for both set of data, or both models. The model or data attached to the left port is presented first in the report, followed by the metrics for the dataset or model attached on the right port.

For example, the following image represents a comparison of results from two clustering models that were built on the same data, but with different parameters.

Result Description	Average Distance to Cluster Center	Average Distance to Other Center	Number of Points	Maximal Distance To Cluster Center
Combined Evaluation	67.804293	227.306621	768	594.954323
Evaluation For Cluster No.0	58.815287	227.524177	603	123.71962
Evaluation For Cluster No.1	100.655022	226.511551	165	594.954323
Combined Evaluation	54.470445	165.172624	768	351.334177
Evaluation For Cluster No.0	94.399003	327.657707	26	351.334177
Evaluation For Cluster No.1	49.247118	163.267603	511	122.671133
Evaluation For Cluster No.2	61.530954	151.098399	231	165.645354

Because this is a clustering model, the evaluation results are different than if you compared scores from two regression models, or compared two classification models. However, the overall presentation is the same.

Metrics

This section describes the metrics returned for the specific types of models supported for use with **Evaluate Model**:

- [classification models](#)
- [regression models](#)
- [clustering models](#)

Metrics for classification models

The following metrics are reported when evaluating classification models. If you compare models, they are ranked by the metric you select for evaluation.

- **Accuracy** measures the goodness of a classification model as the proportion of true results to total cases.
- **Precision** is the proportion of true results over all positive results.
- **Recall** is the fraction of all correct results returned by the model.
- **F-score** is computed as the weighted average of precision and recall between 0 and 1, where the ideal F-score value is 1.
- **AUC** measures the area under the curve plotted with true positives on the y axis and false positives on the x axis. This metric is useful because it provides a single number that lets you compare models of different types.
- **Average log loss** is a single score used to express the penalty for wrong results. It is calculated as the difference between two probability distributions – the true one, and the one in the model.
- **Training log loss** is a single score that represents the advantage of the classifier over a random prediction. The log loss measures the uncertainty of your model by comparing the probabilities it outputs to the known values (ground truth) in the labels. You want to minimize log loss for the model as a whole.

Metrics for regression models

The metrics returned for regression models are generally designed to estimate the amount of error. A model is considered to fit the data well if the difference between observed and predicted values is small. However, looking at the pattern of the residuals (the difference between any one predicted point and its corresponding actual value) can tell you a lot about potential bias in the model.

The following metrics are reported for evaluating regression models. When you compare models, they are ranked by the metric you select for evaluation.

- **Negative log likelihood** measures the loss function, a lower score is better. Note that this metric is only calculated for **Bayesian Linear Regression** and **Decision Forest Regression**; for other algorithms, the value is `Infinity` which means for nothing.
- **Mean absolute error (MAE)** measures how close the predictions are to the actual outcomes; thus, a lower score is better.
- **Root mean squared error (RMSE)** creates a single value that summarizes the error in the model. By squaring the difference, the metric disregards the difference between over-prediction and under-prediction.
- **Relative absolute error (RAE)** is the relative absolute difference between expected and actual values; relative because the mean difference is divided by the arithmetic mean.

- **Relative squared error (RSE)** similarly normalizes the total squared error of the predicted values by dividing by the total squared error of the actual values.

- **Mean Zero One Error (MZOE)** indicates whether the prediction was correct or not. In other words:

$$\text{ZeroOneLoss}(x,y) = 1 \text{ when } x=y ; \text{ otherwise } 0 .$$

- **Coefficient of determination**, often referred to as R^2 , represents the predictive power of the model as a value between 0 and 1. Zero means the model is random (explains nothing); 1 means there is a perfect fit. However, caution should be used in interpreting R^2 values, as low values can be entirely normal and high values can be suspect.

Metrics for clustering models

Because clustering models differ significantly from classification and regression models in many respects, [Evaluate Model](#) also returns a different set of statistics for clustering models.

The statistics returned for a clustering model describe how many data points were assigned to each cluster, the amount of separation between clusters, and how tightly the data points are bunched within each cluster.

The statistics for the clustering model are averaged over the entire dataset, with additional rows containing the statistics per cluster.

For example, the following results show a portion of the results from a sample experiment that clusters the data in the PIMA Indian Diabetes Binary Classification dataset, which is available in Azure Machine Learning Studio (classic).

RESULT DESCRIPTION	AVERAGE DISTANCE TO CLUSTER CENTER	AVERAGE DISTANCE TO OTHER CENTER	NUMBER OF POINTS	MAXIMAL DISTANCE TO CLUSTER CENTER
Combined Evaluation	55.915068	169.897505	538	303.545166
Evaluation For Cluster No.0	0	1	570	0
Evaluation For Cluster No.1	0	1	178	0
Evaluation For Cluster No.2	0	1	178	0

From these results, you get the following information:

- The **Sweep Clustering** module creates multiple clustering models, listed in order of accuracy. For simplicity, we've shown only the best-ranked model here. Models are measured using all possible metrics, but the models are ranked by using the metric that you specified. If you changed the metric, a different model might be ranked higher.
- The **Combined Evaluation** score at the top of the each section of results lists the averaged scores for the clusters created in that particular model.

This top-ranked model happened to create three clusters; other models might create two clusters, or four clusters. Therefore, this combined evaluation score helps you compare models with different number of clusters.

- The scores in the column, **Average Distance to Cluster Center**, represent the closeness of all points in a cluster to the centroid of that cluster.
- The scores in the column, **Average Distance to Other Center**, represent how close, on average, each point in the cluster is to the centroids of all other clusters.

You can choose any one of four metrics to measure this distance, but all measurements must use the same metric.

- The **Number of Points** column shows how many data points were assigned to each cluster, along with the total overall number of data points in any cluster.

If the number of data points assigned to clusters is less than the total number of data points available, it means that the data points could not be assigned to a cluster.

- The scores in the column, **Maximal Distance to Cluster Center**, represent the sum of the distances between each point and the centroid of that point's cluster.

If this number is high, it can mean that the cluster is widely dispersed. You should review this statistic together with the **Average Distance to Cluster Center** to determine the cluster's spread.

Examples

For examples of how to generate, visualize, and interpret evaluation metrics, see these sample experiments in the [Azure AI Gallery](#). These experiments demonstrate how to build multiple models and use **Evaluate Model** to determine which model is the best.

- [Compare Binary Classifiers](#): Explains how to compare the performance of different classifiers that were built using the same data.
- [Compare Multi-class Classifiers](#): Demonstrates how to compare the accuracy of different classification models that were built on the letter recognition dataset.
- [Compare Regressors](#): Walks you through the process of evaluating different regression models.
- [Demand estimation](#): Learn how to combine evaluation metrics from multiple models.
- [Customer relationship prediction](#): Demonstrates how to evaluate multiple related models.

Expected inputs

NAME	TYPE	DESCRIPTION
Scored dataset	Data Table	Scored dataset
<i>Scored dataset to compare</i>	Data Table	Scored dataset to compare (optional)

Outputs

NAME	TYPE	DESCRIPTION
Evaluation results	Data Table	Data evaluation result

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0013	Exception occurs if passed to module learner has invalid type.

EXCEPTION	DESCRIPTION
Error 0020	Exception occurs if number of columns in some of the datasets passed to the module is too small.
Error 0021	Exception occurs if number of rows in some of the datasets passed to the module is too small.
Error 0024	Exception occurs if dataset does not contain a label column.
Error 0025	Exception occurs if dataset does not contain a score column.

See also

[Cross-Validate Model](#)

[Evaluate Recommender](#)

[Evaluate](#)

[Score Model](#)

Evaluate Recommender

3/24/2021 • 9 minutes to read • [Edit Online](#)

Evaluates the accuracy of recommender model predictions

Category: [Machine Learning / Evaluate](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Evaluate Recommender** module in Azure Machine Learning Studio (classic), to measure the accuracy of predictions made by a recommendation model. Using this module, you can evaluate four different kinds of recommendations:

- Ratings predicted for a given user and item
- Items recommended for a given user
- A list of users found to be related to a given user
- A list of items found to be related to a given item

When you create predictions using a recommendation model, slightly different results are returned for each of these supported prediction types. The **Evaluate Recommender** module deduces the kind of prediction from the column format of the scored dataset. For example, the **scored dataset** might contain:

- user-item-rating triples
- users and their recommended items
- users and their related users
- items and their related items

The module also applies the appropriate performance metrics, based on the type of prediction being made.

TIP

Learn everything you need to know about the end-to-end experience of building a recommendation system in this tutorial from the .NET development team. Includes sample code and discussion of how to call Azure Machine Learning from an application.

[Building recommendation engine for .NET applications using Azure Machine Learning](#)

How to configure Evaluate Recommender

The **Evaluate Recommender** module compares the predictions output by a recommendation model with the corresponding "ground truth" data. For example, the **Score Matchbox Recommender** module produces scored datasets that can be analyzed with **Evaluate Recommender**.

Requirements

Evaluate Recommender requires the following datasets as input.

Test dataset

The **test dataset** contains the "ground truth" data in the form of **user-item-rating triples**.

If you already have a dataset containing user-item-rating triples, you can apply the [Split Data](#) module, using the **RecommenderSplit** option, to create a training dataset and a related test set from the existing dataset.

Scored dataset

The **scored dataset** contains the predictions that were generated by the recommendation model.

The columns in this second dataset depend on the kind of prediction you were performing during scoring. For example, the scored dataset might contain any of the following:

- Users, items, and the ratings the user would likely give for the item
- A list of users and items recommended for them
- A list of users, with users who are probably similar to them
- A list of items, together with similar items

Metrics

Performance metrics for the model are generated based on the type of input. For details, see these sections:

- [Evaluate predicted ratings](#)
- [Evaluate item recommendations](#)
- [Evaluate predictions of related users](#)
- [Evaluate predictions of related items](#)

Evaluate predicted ratings

When evaluating predicted ratings, the scored dataset (the second input to **Evaluate Recommender**) must contain **user-item-rating triples**, meeting these requirements:

- The first column of the dataset contains user identifiers.
- The second column contains the item identifiers.
- The third column contains the corresponding user-item ratings.

IMPORTANT

For evaluation to succeed, the column names must be `User`, `Item`, and `Rating`, respectively.

Evaluate Recommender compares the ratings in the ground truth dataset to the predicted ratings of the scored dataset, and computes the **mean absolute error** (MAE) and the **root mean squared error** (RMSE).

The other parameters of **Evaluate Recommender** have no effect on evaluation of rating predictions.

Evaluate item recommendations

When evaluating item recommendation, use a scored dataset that includes the recommended items for each user:

- The first column of the dataset must contain the user identifier.
- All subsequent columns should contain the corresponding recommended item identifiers, ordered by how relevant an item is to the user.

Before connecting this dataset, we recommend that you sort the dataset so that the most relevant items come first.

The other parameters of **Evaluate Recommender** have no effect on evaluation of item recommendations.

IMPORTANT

For **Evaluate Recommender** to work, the column names must be `User`, `Item 1`, `Item 2`, `Item 3` and so forth.

Evaluate Recommender computes the average normalized discounted cumulative gain (NDCG) and returns it in the output dataset.

Because it is impossible to know the actual "ground truth" for the recommended items, **Evaluate Recommender** uses the user-item ratings in the test dataset as gains in the computation of the NDCG. To evaluate, the recommender scoring module must only produce recommendations for items with ground truth ratings (in the test dataset).

Evaluate predictions of related users

When evaluating predictions of related users, use a scored dataset that contains the related users for each user of interest:

- The first column must contain the identifiers for each user of interest.
- All subsequent columns contain the identifiers for the predicted related users. Related users are ordered by the strength of the relationship (most related user first).
- For **Evaluate Recommender** to work, the column names must be `User`, `Related User 1`, `Related User 2`, `Related User 3`, and so forth.

TIP

You can influence evaluation by setting the minimum number of items that a user of interest and its related users must have in common.

Evaluate Recommender computes the average normalized discounted cumulative gain (NDCG), based on Manhattan (L1 Sim NDCG) and Euclidean (L2 Sim NDCG) distances, and returns both values in the output dataset. Because there is no actual ground truth for the related users, **Evaluate Recommender** uses the following procedure to compute the average NDCGs.

For each user of interest in the scored dataset:

1. Find all items in the test dataset which have been rated by both the user of interest and the related user under consideration.
2. Create two vectors from the ratings of these items: one for the user of interest, and one for the related user under consideration.
3. Compute the gain as the similarity of the resulting two rating vectors, in terms of their Manhattan (L1) or Euclidean (L2) distance.
4. Compute the L1 Sim NDCG and the L2 Sim NDCG, using the gains of all related users.
5. Average the NDCG values over all users in the scored dataset.

In other words, gain is computed as the similarity (normalized Manhattan or Euclidian distances) between a user of interest (the entry in the first column of scored dataset) and a given related user (the entry in the n-th column

of the scored dataset). The gain of this user pair is computed using all items for which both items have been rated in the original data (test set). The NDCG is then computed by aggregating the individual gains for a single user of interest and all related users, using logarithmic discounting. That is, one NDCG value is computed for each user of interest (each row in the scored dataset). The number that is finally reported is the arithmetic average over all users of interest in the scored dataset (i.e. its rows).

Hence, to evaluate, the recommender scoring module must only predict related users who have items with ground truth ratings (in the test dataset).

Evaluate predictions of related items

When evaluating the prediction of related items, use a scored dataset that contains the related items for each item of interest:

- The first column must contain identifiers for the items of interest.
- All subsequent columns should contain identifiers for the predicted related items, ordered by how related they are to the item of interest (most related item first).
- For **Evaluate Recommender** to work, the column names must be `Item`, `Related Item 1`, `Related Item 2`, `Related Item 3`, and so forth.

TIP

You can influence evaluation by setting the minimum number of users that an item of interest and its related items must have in common.

Evaluate Recommender computes the average normalized discounted cumulative gain (**NDCG**) based on Manhattan (**L1 Sim NDCG**) and Euclidean (**L2 Sim NDCG**) distances and returns both values in the output dataset. Because there is no actual ground truth for the related items, **Evaluate Recommender** computes the average NDCGs as follows:

For each item of interest in the scored dataset:

1. Find all users in the test dataset who have rated both the item of interest and the related item under consideration.
2. Create two vectors from the ratings of these users, one for the item of interest and for the related item under consideration.
3. Compute the gain as the similarity of the resulting two rating vectors in terms of their Manhattan (L1) or Euclidean (L2) distance.
4. Compute the L1 Sim NDCG and the L2 Sim NDCG using the gains of all related items.
5. Average the NDCG values over all items of interest in the scored dataset.

In other words, gain is computed as the similarity (normalized Manhattan or Euclidian distances) between an item of interest (the entry in the first column of scored dataset) and a given related item (the entry in the n-th column of the scored dataset). The gain of this item pair is computed using all users who have rated both of these items in the original data (test set). The NDCG is then computed by aggregating the individual gains for a single item of interest and all its related items, using logarithmic discounting. That is, one NDCG value is computed for each item of interest (each row in the scored dataset). The number that is finally reported is the arithmetic average over all items of interest in the scored dataset (i.e. its rows).

Therefore, to evaluate, the recommender scoring module must only predict related items with ground truth ratings (in the test dataset).

Examples

For examples of how recommendation models are used in Azure Machine Learning, see the [Azure AI Gallery](#):

- [Movie recommender sample](#): Demonstrates how to train, evaluate, and score using a recommendation model.
- [Building recommendation engine for .NET applications using Azure Machine Learning](#): This blog provides a detailed description of how to build a movie recommendation model.

Expected inputs

NAME	TYPE	DESCRIPTION
Test dataset	Data Table	Test dataset
Scored dataset	Data Table	Scored dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Minimum number of items that the query user and the related user must have rated in common	>=1	Integer	2	Specify the minimum number of items that must have been rated by both the query user and the related user This parameter is optional
Minimum number of users that the query item and the related item must have been rated by in common	>=1	Integer	2	Specify the minimum number of users that must have rated both the query item and the related item This parameter is optional

Outputs

NAME	TYPE	DESCRIPTION
Metric	Data Table	A table of evaluation metrics

Exceptions

EXCEPTION	DESCRIPTION
Error 0022	Exception occurs if number of selected columns in input dataset does not equal to the expected number.

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.
Error 0034	Exception occurs if more than one rating exists for a given user-item pair.
Error 0018	Exception occurs if input dataset is not valid.
Error 0002	Exception occurs if one or more parameters could not be parsed or converted from specified type into required by target method type.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Train Matchbox Recommender](#)

[Score Matchbox Recommender](#)

Machine Learning - Initialize Model

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that you can use to define a machine learning model and set its parameters.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

You can think of the *untrained model* as a specification that you can apply to different input datasets. You might apply the same model specification to different data and get different results. Or, you can use the specification to retrain a model. You can then add new data.

This article also describes the overall process of creating, training, evaluating, and scoring a model in Machine Learning Studio (classic).

Create and use machine learning models in Machine Learning Studio (classic)

The typical workflow for machine learning includes these phases:

- Choose a suitable algorithm and set initial options.
- Train the model by using compatible data.
- Create predictions by using new data based on the patterns in the model.
- Evaluate the model to determine whether the predictions are accurate, the amount of error, and whether overfitting occurs.

Machine Learning Studio (classic) supports a flexible, customizable framework for machine learning. Each task in this process is performed by a specific type of module. Modules can be modified, added, or removed without breaking the rest of your experiment.

Use the modules in this category to select an initial algorithm. Then, configure detailed parameters based on the specific model type. You can then apply this model specification to a set of data.

About creating models

Azure Machine Learning provides many state-of-the art machine learning algorithms to help you build analytical models. Each algorithm is packaged in its own module. To create a customized model:

1. Choose a model by category.

Algorithms are grouped by specific types of predictive tasks. Examples include regression, classification, and image recognition. Your first task is to identify the general category of machine learning task to perform, and then to select an algorithm. If you need help selecting an algorithm, see these resources:

- [Machine learning algorithm cheat sheet for Machine Learning Studio \(classic\)](#)
- [Choose Azure Machine Learning algorithms for clustering, classification, or regression](#)

2. Configure algorithm parameters.

Use the **Properties** pane in each module to set parameters. Parameters control how the model learns from data.

3. Train the model on data.

After you configure the model, connect a dataset. Then, use one of the [training modules](#) to run data through the algorithms that you want to use.

You can use [Tune Model Hyperparameters](#) to iterate over all possible parameters and determine the optimal configuration for your task and data.

4. Predict, score, or evaluate.

After you build and train a model, typically your next step is to use one of the [scoring modules](#) to generate predictions based on the model.

You can use the modules for [model evaluation](#) to measure the accuracy of the model based on the scores that you generate.

List of modules

The modules in this category are organized by the type of machine learning algorithm that the modules encapsulate. Each type of algorithm typically requires a different type of data.

- [Anomaly Detection](#)
- [Classification](#)
- [Clustering](#)
- [Regression](#)

Related tasks

In addition to the traditional machine learning algorithm categories described here, the following modules provide specialized types of learning from data or preprocessing:

- [Text analysis](#)
- [Image classification](#)
- [Time series analysis](#)

See also

- [Train](#)
- [Score](#)
- [Evaluate](#)
- [Module categories and descriptions](#)

Anomaly Detection

3/10/2021 • 2 minutes to read • [Edit Online](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

This article introduces the modules provided in Azure Machine Learning Studio (classic) for anomaly detection. Anomaly detection encompasses many important tasks in machine learning:

- Identifying transactions that are potentially fraudulent.
- Learning patterns that indicate that a network intrusion has occurred.
- Finding abnormal clusters of patients.
- Checking values entered into a system.

Because anomalies are rare events by definition, it can be difficult to collect a representative sample of data to use for modeling. The algorithms included in this category have been especially designed to address the core challenges of building and training models by using imbalanced data sets.

Anomaly detection modules

Machine Learning Studio (classic) provides the following modules that you can use to create an anomaly detection model. Just drag the module into your experiment to begin working with the model.

- [One-Class Support Vector Machine](#)
- [PCA-Based Anomaly Detection](#)

After setting model parameters, you must train the model by using a labeled data set and the [Train Anomaly Detection Model](#) training module. The result is a trained model that you can use to test new data. To do this, use the all-purpose [Score Model](#) module.

For an example of how these modules work together, see the [Anomaly Detection: Credit Risk](#) experiment in the Cortana Intelligence Gallery.

Related tasks

[Time Series Anomaly Detection](#) is a new module that's a bit different from the other anomaly detection models. The Time Series Anomaly Detection module is designed for time series data. It's intended to use to analyze trends over time. The algorithm identifies potentially anomalous trends in the time series data. It flags deviations from the trend's direction or magnitude.

Azure also provides the [Machine Learning Anomaly Detection API](#), which you can call as a web service.

TIP

If you're not sure whether anomaly detection is the right algorithm to use with your data, see these guides:

- [Machine learning algorithm cheat sheet for Azure Machine Learning](#) provides a graphical decision chart to guide you through the selection process.
- [Choose Azure Machine Learning algorithms for clustering, classification, or regression.](#)

List of modules

The Anomaly Detection category includes the following modules:

- [One-Class Support Vector Machine](#): Creates a one-class support vector machine model for anomaly detection.
- [PCA-Based Anomaly Detection](#): Creates an anomaly detection model by using Principal Component Analysis.

See also

- [Regression](#)
- [Classification](#)
- [Clustering](#)
- [Text Analytics](#)
- [OpenCV Library Modules](#)
- [Module categories and descriptions](#)

One-Class Support Vector Machine

3/10/2021 • 5 minutes to read • [Edit Online](#)

Creates a one class Support Vector Machine model for anomaly detection

Category: [Anomaly Detection](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **One-Class Support Vector Model** module in Azure Machine Learning, to create an anomaly detection model.

This module is particularly useful in scenarios where you have a lot of "normal" data and not many cases of the anomalies you are trying to detect. For example, if you need to detect fraudulent transactions, you might not have many examples of fraud that you could use to train a typical classification model, but you might have many examples of good transactions.

You use the **One-Class Support Vector Model** module to create the model, and then train the model using the [Train Anomaly Detection Model](#). The dataset that you use for training can contain all or mostly normal cases.

You can then apply different metrics to identify potential anomalies. For example, you might use a large dataset of good transactions to identify cases that possibly represent fraudulent transactions.

More about one-class SVM

Support vector machines (SVMs) are supervised learning models that analyze data and recognize patterns, and that can be used for both classification and regression tasks.

Typically, the SVM algorithm is given a set of training examples labeled as belonging to one of two classes. An SVM model is based on dividing the training sample points into separate categories by as wide a gap as possible, while penalizing training samples that fall on the wrong side of the gap. The SVM model then makes predictions by assigning points to one side of the gap or the other.

Sometimes oversampling is used to replicate the existing samples so that you can create a two-class model, but it is impossible to predict all the new patterns of fraud or system faults from limited examples. Moreover, collection of even limited examples can be expensive.

Therefore, in one-class SVM, the support vector model is trained on data that has only one class, which is the "normal" class. It infers the properties of normal cases and from these properties can predict which examples are unlike the normal examples. This is useful for anomaly detection because the scarcity of training examples is what defines anomalies: that is, typically there are very few examples of the network intrusion, fraud, or other anomalous behavior.

For more information, including links to basic research, see the [Technical notes](#) section.

NOTE

The One-Class Support Vector Model module creates a kernel-SVM model, which means that it is not very scalable. If training time is limited, or you have too much data, you can use other methods for anomaly detectors, such as [PCA-Based Anomaly Detection](#).

How to configure One-Class SVM

1. Add the One-Class Support Vector Model module to your experiment in Studio (classic). You can find the module under **Machine Learning - Initialize**, in the **Anomaly Detection** category.
2. Double-click the One-Class Support Vector Model module to open the **Properties** pane.
3. For **Create trainer mode**, select an option that indicates how the model should be trained:
 - **Single Parameter**: Use this option if you know how you want to configure the model, and provide a specific set of values as arguments.
 - **Parameter Range**: Use this option if you are not sure of the best parameters, and want to perform a parameter sweep to find the optimal configuration.
4. **η** : Type a value that represents the upper bound on the fraction of outliers. This parameter corresponds to the nu-property described in [this paper](#). The nu-property lets you control the trade-off between outliers and normal cases.
5. **ϵ (epsilon)**: Type a value to use as the stopping tolerance. The stopping tolerance, affects the number of iterations used when optimizing the model, and depends on the stopping criterion value. When the value is exceeded, the trainer stops iterating on a solution.
6. Connect a training dataset, and one of the training modules:
 - If you set **Create trainer mode** to **Single Parameter**, use the [Train Anomaly Detection Model](#) module.
 - If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#) module.

NOTE

If you pass a parameter range to [Train Anomaly Detection Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value is used throughout the sweep, even if other parameters change across a range of values.

7. Run the experiment.

Results

The module returns a trained anomaly detection model. You can either save the model in your workspace, or you can connect the [Score Model](#) module and use the trained model to detect possible anomalies.

If you trained the model using a parameter sweep, make a note of the optimal parameter settings to use when configuring a model for use in production.

Examples

For examples of how this module is used in anomaly detection, see the [Azure AI Gallery](#):

- [Anomaly Detection: Credit Risk](#): This sample illustrates how to find outliers in data, using a parameter sweep to find the optimal model. It then applies that model to new data to identify risky transactions that might represent fraud, comparing two different anomaly detection models.

Technical notes

Predictions from the [One-Class SVM](#) are uncalibrated scores that may be possibly unbounded. As the example in the Cortana Intelligence Gallery demonstrates, be sure to normalize scores if you are comparing models based on different algorithms.

Research

This implementation wraps the library for support vector machines named [libsvm](#). The general theory on which [libsvm](#) is based, and the approach towards one-class support vector machines, is described in these papers by B. Schölkopf et al.

- [Estimating the Support of a High-Dimensional Distribution](#)
- [New Support Vector Algorithms](#)

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DESCRIPTION	DEFAULT
Create trainer mode	Create Trainer Mode	List:Single Parameter Parameter Range	Required	Single Parameter Specify learner options. Use the SingleParameter option to manually specify all values. Use the ParameterRange option to sweep over tunable parameters.	
nu	Float	>=double.Epsilon	mode:Single Parameter	0.1	This parameter (represented by the Greek letter nu) determines the trade-off between the fraction of outliers and the number of support vectors.
epsilon	Float	>=double.Epsilon	mode:Single Parameter	0.001	Specifies the stopping tolerance.

NAME	TYPE	RANGE	OPTIONAL	DESCRIPTION	DEFAULT
psnu	ParameterRange Settings	[0.001;1.0]	mode:Parameter Range	0.001; 0.01; 0.1	Specifies the range for the trade-off between the fraction of outliers and the number of support vectors.
psEpsilon	ParameterRange Settings	[1e-6;1.0]	mode:Parameter Range	0.001; 0.01; 0.1	Specifies the range for stopping tolerance.

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained anomaly detection model

See also

[Classification](#)

[Train Anomaly Detection Model](#)

PCA-Based Anomaly Detection

3/10/2021 • 8 minutes to read • [Edit Online](#)

Creates an anomaly detection model using Principal Component Analysis

Category: [Anomaly Detection](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **PCA-Based Anomaly Detection** module in Azure Machine Learning Studio (classic), to create an anomaly detection model based on Principal Component Analysis (PCA).

This module helps you build a model in scenarios where it is easy to obtain training data from one class, such as valid transactions, but difficult to obtain sufficient samples of the targeted anomalies.

For example, to detect fraudulent transactions, very often you don't have enough examples of fraud to train on, but have many examples of good transactions. The **PCA-Based Anomaly Detection** module solves the problem by analyzing available features to determine what constitutes a "normal" class, and applying distance metrics to identify cases that represent anomalies. This let you train a model using existing imbalanced data.

More about Principal Component Analysis

Principal Component Analysis, which is frequently abbreviated to PCA, is an established technique in machine learning. PCA is frequently used in exploratory data analysis because it reveals the inner structure of the data and explains the variance in the data.

PCA works by analyzing data that contains multiple variables. It looks for correlations among the variables and determines the combination of values that best captures differences in outcomes. These combined feature values are used to create a more compact feature space called the *principal components*.

For anomaly detection, each new input is analyzed, and the anomaly detection algorithm computes its projection on the eigenvectors, together with a normalized reconstruction error. The normalized error is used as the anomaly score. The higher the error, the more anomalous the instance is.

For additional information about how PCA works, and about the implementation for anomaly detection, see these papers:

- [A randomized algorithm for principal component analysis](#). Rokhlin, Szlan and Tygert
- [Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions](#) (PDF download). Halko, Martinsson and Tropp.

How to configure PCA Anomaly Detection

1. Add the **PCA-Based Anomaly Detection** module to your experiment in Studio (classic). You can find this module under **Machine Learning**, **Initialize Model**, in the **Anomaly Detection** category.

2. In the **Properties** pane for the PCA-Based Anomaly Detection module, click the **Training mode** option, and indicate whether you want to train the model using a specific set of parameters, or use a parameter sweep to find the best parameters.

- **Single Parameter:** Select this option if you know how you want to configure the model, and provide a specific set of values as arguments.
- **Parameter Range:** Select this option if you are not sure of the best parameters and want to use a parameter sweep, using the [Tune Model Hyperparameters](#) module. The trainer iterates over a range of settings you specify, and determines the combination of settings that produces the optimal results.

3. **Number of components to use in PCA, Range for number of PCA components:** Specify the number of output features, or components, that you want to output.

The decision of how many components to include is an important part of experiment design using PCA. General guidance is that you should not include the same number of PCA components as there are variables. Instead, you should start with some smaller number of components and increase them until some criteria is met.

If you are unsure of what the optimum value might be, we recommend that you train the anomaly detection model using the **Parameter Range** option.

The best results are obtained when the number of output components is **less than** the number of feature columns available in the dataset.

4. Specify the amount of oversampling to perform during randomized PCA training. In anomaly detection problems, imbalanced data makes it difficult to apply standard PCA techniques. By specifying some amount of oversampling, you can increase the number of target instances.

If you specify 1, no oversampling is performed. If you specify any value higher than 1, additional samples are generated to use in training the model.

There are two options, depending on whether you are using a parameter sweep or not:

- **Oversampling parameter for randomized PCA:** Type a single whole number that represents the ratio of oversampling of the minority class over the normal class. (Available when using the **Single parameter** training method.)
- **Range for the oversampling parameter used in randomized PCA:** Type a series of numbers to try, or use the Range Builder to select values using a slider. (Available only when using the **Parameter range** training method.)

NOTE

You cannot view the oversampled data set. For additional details of how oversampling is used with PCA, see [Technical notes](#).

5. **Enable input feature mean normalization:** Select this option to normalize all input features to a mean of zero. Normalization or scaling to zero is generally recommended for PCA, because the goal of PCA is to maximize variance among variables.

This option is selected by default. Deselect this option if values have already been normalized using a different method or scale.

6. Connect a tagged training dataset, and one of the training modules:

- If you set the **Create trainer mode** option to **Single Parameter**, use the [Train Anomaly Detection Model](#) module.

- If you set the **Create trainer mode** option to **Parameter Range**, use the [Tune Model Hyperparameters](#) module.

NOTE

If you pass a parameter range to [Train Anomaly Detection Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and using the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value is used throughout the sweep, even if other parameters change across a range of values.

7. Run the experiment, or select the module and click **Run selected**.

Results

When training is complete, you can either save the trained model, or connect it to the [Score Model](#) module to predict anomaly scores.

To evaluate the results of an anomaly detection models requires some additional steps:

1. Ensure that a score column is available in both datasets

If you try to evaluate an anomaly detection model and get the error, "There is no score column in scored dataset to compare", it means you are using a typical evaluation dataset that contains a label column but no probability scores. You need to choose a dataset that matches the schema output for anomaly detection models, which includes a **Scored Labels** and **Scored Probabilities** column.

2. Ensure that label columns are marked

Sometimes the metadata associated with the label column is removed in the experiment graph. If this happens, when you use the [Evaluate Model](#) module to compare the results of two anomaly detection models, you might get the error, "There is no label column in scored dataset", or "There is no label column in scored dataset to compare".

You can avoid this error by adding the [Edit Metadata](#) module before the [Evaluate Model](#) module. Use the column selector to choose the class column, and in the **Fields** dropdown list, select **Label**.

3. Normalize scores from different model types

Predictions from the PCA anomaly detection model always are in the range [0,1]. In contrast, output from the [One-Class SVM](#) module are uncalibrated scores that are possibly unbounded.

Therefore, if you are comparing models based on different algorithms, you must always normalize scores. See the example in the Azure AI Gallery for an example of normalization among different anomaly detection models.

Examples

For examples of how PCA is used in anomaly detection, see the [Azure AI Gallery](#):

- [Anomaly detection: credit risk](#): Illustrates how to find outliers in data. This example uses a parameter sweep to find the optimal model. It then applies that model to new data to identify risky transactions that might represent fraud, comparing two different anomaly detection models.

Technical notes

This algorithm uses PCA to approximate the subspace containing the normal class. The subspace is spanned by eigenvectors associated with the top eigenvalues of the data covariance matrix. For each new input, the anomaly detector first computes its projection on the eigenvectors, and then computes the normalized reconstruction error. This error is the anomaly score. The higher the error, the more anomalous the instance. For details on how the normal space is computed, see Wikipedia: [Principal Component Analysis](#)

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DESCRIPTION	DEFAULT
Training mode	CreateLearnerMode	List:Single Parameter Parameter Range	Required	Single Parameter Specify learner options. Use the SingleParameter option to manually specify all values.	
Number of components to use in PCA	Integer		mode:Single Parameter	2	Specify the number of components to use in PCA.
Oversampling parameter for randomized PCA	Integer		mode:Single Parameter	2	Specify the accuracy parameter for randomized PCA training.
Enable input feature mean normalization	Logic type	List:True False	Required	False	Specify if the input data is normalized to have zero mean.
Range for number of PCA components	ParameterRange Settings	[1;100]	mode:Parameter Range	2; 4; 6; 8; 10	Specify the range for number of components to use in PCA.
Range for the oversampling parameter used in randomized PCA	ParameterRange Settings	[1;100]	mode:Parameter Range	2; 4; 6; 8; 10	Specify the range for accuracy parameter used in randomized PCA training.

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained PCA-based anomaly detection model

Exceptions

EXCEPTION	DESCRIPTION
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.
Error 0062	Exception occurs when attempting to compare two models with different learner types.
Error 0047	Exception occurs if number of feature columns in some of the datasets passed to the module is too small.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[One-Class Support Vector Machine](#)

Classification modules

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that support the creation of classification models. You can use these modules to build binary or multiclass classification models.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

About classification

Classification is a machine learning method that uses data to determine the category, type, or class of an item or row of data. For example, you can use classification to:

- Classify email filters as spam, junk, or good.
- Determine whether a patient's lab sample is cancerous.
- Categorize customers by their propensity to respond to a sales campaign.
- Identify sentiment as positive or negative.

Classification tasks are frequently organized by whether a classification is binary (either A or B) or multiclass (multiple categories that can be predicted by using a single model).

Create a classification model

To create a classification model, or *classifier*, first, select an appropriate algorithm. Consider these factors:

- How many classes or different outcomes do you want to predict?
- What is the distribution of the data?
- How much time can you allow for training?

Machine Learning Studio (classic) provides multiple classification algorithms. When you use the [One-Vs-All](#) algorithm, you can even apply a binary classifier to a multiclass problem.

After you choose an algorithm and set the parameters by using the modules in this section, train the model on labeled data. Classification is a supervised machine learning method. It always requires labeled training data.

When training is finished, you can [evaluate](#) and tune the model. When you're satisfied with the model, use the trained model for [scoring](#) with new data.

List of modules

The **Classification** category includes the following modules:

- [Multiclass Decision Forest](#): Creates a multiclass classification model by using the decision forest algorithm.
- [Multiclass Decision Jungle](#): Creates a multiclass classification model by using the decision jungle algorithm.
- [Multiclass Logistic Regression](#): Creates a multiclass logistic regression classification model.
- [Multiclass Neural Network](#): Creates a multiclass classification model by using a neural network algorithm.
- [One-vs-All Multiclass](#): Creates a multiclass classification model from an ensemble of binary classification

models.

- [Two-Class Averaged Perceptron](#): Creates an averaged perceptron binary classification model.
- [Two-Class Bayes Point Machine](#): Creates a Bayes point machine binary classification model.
- [Two-Class Boosted Decision Tree](#): Creates a binary classifier by using a boosted decision tree algorithm.
- [Two-Class Decision Forest](#): Creates a two-class classification model by using the decision forest algorithm.
- [Two-Class Decision Jungle](#): Creates a two-class classification model by using the decision jungle algorithm.
- [Two-Class Locally Deep Support Vector Machine](#): Creates a binary classification model by using the locally deep Support Vector Machine algorithm.
- [Two-Class Logistic Regression](#): Creates a two-class logistic regression model.
- [Two-Class Neural Network](#): Creates a binary classifier by using a neural network algorithm.
- [Two-Class Support Vector Machine](#): Creates a binary classification model by using the Support Vector Machine algorithm.

Examples

For examples of classification in action, see the [Azure AI Gallery](#).

For help choosing an algorithm, see these articles:

- [Machine learning algorithm cheat sheet for Azure Machine Learning](#)

Provides a graphical decision chart to guide you through the selection process.

- [Choose Azure Machine Learning algorithms for clustering, classification, or regression](#)

Explains in greater detail the different types of machine learning algorithms and how they're used.

See also

- [Regression](#)
- [Clustering](#)
- [Text Analytics](#)
- [Image classification](#)
- [A-Z module list](#)

Multiclass Decision Forest

3/24/2021 • 7 minutes to read • [Edit Online](#)

Creates a multiclass classification model using the decision forest algorithm

Category: [Machine Learning / Initialize Model / Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Multiclass Decision Forest** module in Azure Machine Learning Studio (classic), to create a machine learning model based on the *decision forest* algorithm. A decision forest is an ensemble model that very rapidly builds a series of decision trees, while learning from tagged data.

More about decision forests

The decision forest algorithm is an ensemble learning method for classification. The algorithm works by building multiple decision trees and then *voting* on the most popular output class. Voting is a form of aggregation, in which each tree in a classification decision forest outputs a non-normalized frequency histogram of labels. The aggregation process sums these histograms and normalizes the result to get the “probabilities” for each label. The trees that have high prediction confidence have a greater weight in the final decision of the ensemble.

Decision trees in general are non-parametric models, meaning they support data with varied distributions. In each tree, a sequence of simple tests is run for each class, increasing the levels of a tree structure until a leaf node (decision) is reached.

Decision trees have many advantages:

- They can represent non-linear decision boundaries.
- They are efficient in computation and memory usage during training and prediction.
- They perform integrated feature selection and classification.
- They are resilient in the presence of noisy features.

The decision forest classifier in Azure Machine Learning Studio (classic) consists of an ensemble of decision trees. Generally, ensemble models provide better coverage and accuracy than single decision trees. For more information, see [Decision trees](#).

How to configure Multiclass Decision Forest

TIP

If you are not sure of the best parameters, we recommend that you use the [Tune Model Hyperparameters](#) module to train and test multiple models and find the optimal parameters.

1. Add the **Multiclass Decision Forest** module to your experiment in Studio (classic). You can find this module under **Machine Learning**, **Initialize Model**, and **Classification**.
2. Double-click the module to open the **Properties** pane.
3. For **Resampling method**, choose the method used to create the individual trees. You can choose from bagging or replication.
 - **Bagging:** Bagging is also called *bootstrap aggregating*. In this method, each tree is grown on a new sample, created by randomly sampling the original dataset with replacement until you have a dataset the size of the original. The outputs of the models are combined by *voting*, which is a form of aggregation. For more information, see the Wikipedia entry for Bootstrap aggregating.
 - **Replicate:** In replication, each tree is trained on exactly the same input data. The determination of which split predicate is used for each tree node remains random, creating diverse trees.

See the [How to Configure a Multiclass Decision Forest Model](#) section for guidance.

4. Specify how you want the model to be trained, by setting the **Create trainer mode** option.
 - **Single Parameter:** Select this option if you know how you want to configure the model, and provide a set of values as arguments.
 - **Parameter Range:** Use this option if you are not sure of the best parameters, and want to use a parameter sweep.
5. **Number of decision trees:** Type the maximum number of decision trees that can be created in the ensemble. By creating more decision trees, you can potentially get better coverage, but training time might increase.

This value also controls the number of trees displayed in the results, when visualizing the trained model. To see or print a single tree, you can set the value to 1; however, this means that only one tree can be produced (the tree with the initial set of parameters), and no further iterations are performed.
6. **Maximum depth of the decision trees:** Type a number to limit the maximum depth of any decision tree. Increasing the depth of the tree might increase precision, at the risk of some overfitting and increased training time.
7. **Number of random splits per node:** Type the number of splits to use when building each node of the tree. A *split* means that features in each level of the tree (node) are randomly divided.
8. **Minimum number of samples per leaf node:** Indicate the minimum number of cases that are required to create any terminal node (leaf) in a tree. By increasing this value, you increase the threshold for creating new rules.

For example, with the default value of 1, even a single case can cause a new rule to be created. If you increase the value to 5, the training data would have to contain at least 5 cases that meet the same conditions.

9. **Allow unknown values for categorical features:** Select this option to create a group for unknown values in the training or validation sets. The model might be less precise for known values, but it can provide better predictions for new (unknown) values.
- If you deselect this option, the model can accept only the values that are present in the training data.

10. Connect a labeled dataset, and one of the training modules:

- If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
- If you set **Create trainer mode** option to **Parameter Range**, use the [Tune Model](#)

[Hyperparameters](#) module. With this option, the trainer can iterate over multiple combinations of the settings and determine the parameter values that produce the best model.

NOTE

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

11. Run the experiment.

Results

After training is complete:

- To see the tree that was created on each iteration, right-click [Train Model](#) module and select **Trained model** to visualize. If you use [Tune Model Hyperparameters](#), right click the module and select **Trained best model** to visualize the best model. To see the rules for each node, click each tree to drill down into the splits.

Examples

For examples of how decision forests are used in machine learning, see the [Azure AI Gallery](#):

- [Compare Multiclass Classifiers sample](#): Uses several algorithms and discusses their pros and cons.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

Each tree in a classification decision forest outputs an un-normalized frequency histogram of labels. The aggregation is to sum these histograms and normalize to get the “probabilities” for each label. In this manner, the trees that have high prediction confidence have a greater weight in the final decision of the ensemble.

Related research

For more information about the training process with the **Replicate** option, see:

- [Decision forests for computer vision and medical image analysis. Criminisi and Shotton. Springer 2013.](#)

How to Configure a Multiclass Decision Forest Model

You can change the way the module is configured to accommodate scenarios such as too little data, or limited time for training.

Limited training time

If the training set contains a large number of instances, but the time you have available for training the model is limited, try using these options:

- Create a decision forest that uses a smaller number of decision trees (for example, 5-10).
- Use the **Replicate** option for resampling.
- Specify a smaller number of random splits per node (for example, less than 100).

Limited training set

If the training set contains a limited number of instances, try using these options:

- Create a decision forest that uses a large number of decision trees (for example, more than 20).
- Use the **Bagging** option for resampling.
- Specify a large number of random splits per node (for example, more than 1,000).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Resampling method	Any	ResamplingMethod	Bagging	Choose a resampling method: Bagging or Replicate
Number of decision trees	≥ 1	Integer	8	Specify the number of decision trees to create in the ensemble
Maximum depth of the decision trees	≥ 1	Integer	32	Specify the maximum depth of any decision tree that can be created
Number of random splits per node	≥ 1	Integer	128	Specify the number of splits generated per node, from which the optimal split is selected
Minimum number of samples per leaf node	≥ 1	Integer	1	Specify the minimum number of training samples required to generate a leaf node
Allow unknown values for categorical features	Any	Boolean	True	Indicate whether unknown values of existing categorical features can be mapped to a new, additional feature

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained multiclass classification model

See also

[Classification](#)

[Two-Class Decision Forest](#)

[Decision Forest Regression](#)

[A-Z Module List](#)

Multiclass Decision Jungle

3/24/2021 • 5 minutes to read • [Edit Online](#)

Creates a multiclass classification model using the decision jungle algorithm

Category: [Machine Learning](#) / [Initialize Model](#) / [Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Multiclass Decision Jungle** module in Azure Machine Learning Studio (classic), to create a machine learning model that is based on a supervised learning algorithm called *decision jungles*.

You define the model and its parameters using this module, and then connect a labeled training data set to train the model using one of the [training modules](#). The trained model can be used to predict a target that has multiple values.

More about decision jungles

[Decision jungles](#) are a recent extension to [decision forests](#). A decision jungle consists of an ensemble of decision directed acyclic graphs (DAGs).

Decision jungles have the following advantages:

- By allowing tree branches to merge, a decision DAG typically has a lower memory footprint and a better generalization performance than a decision tree, albeit at the cost of a somewhat higher training time.
- Decision jungles are non-parametric models, which can represent non-linear decision boundaries.
- They perform integrated feature selection and classification and are resilient in the presence of noisy features.

For more information about the research behind this machine learning algorithm, see [Decision Jungles: Compact and Rich Models for Classification](#) (downloadable PDF).

How to configure Multiclass Decision Jungle Model

1. Add the **Multiclass Decision Jungle** module to your experiment in Studio (classic). You can find this module under [Machine Learning](#), [Initialize Model](#), and [Classification](#).
2. Double-click the module to open the **Properties** pane.
3. **Resampling method**, choose the method for creating multiple trees, either bagging or replication.
 - **Bagging**: Select this option to use bagging, also called bootstrap aggregating.

Each tree in a decision forest outputs a Gaussian distribution by way of prediction. The

aggregation is to find a Gaussian whose first two moments match the moments of the mixture of Gaussians given by combining all Gaussians returned by individual trees.

- **Replicate:** Select this option to use replication. In this method, each tree is trained on exactly the same input data. The determination of which split predicate is used for each tree node remains random, so diverse trees are created.

4. Specify how you want the model to be trained, by setting the **Create trainer mode** option.

- **Single Parameter:** Use this option when you know how you want to configure the model.
- **Parameter Range:** Use this option if you are not sure of the best parameters, and want to use a parameter sweep.

5. **Number of decision DAGs:** Indicate the maximum number of graphs that can be created in the ensemble.

6. **Maximum depth of the decision DAGs:** Specify the maximum depth of each graph.

7. **Maximum width of the decision DAGs:** Specify the maximum width of each graph.

8. **Number of optimization steps per decision DAG layer:** Indicate how many iterations over the data to perform when building each DAG.

9. **Allow unknown values for categorical features:** Select this option to create a group for unknown values in testing or validation data. The model might be less precise for known values, but it can provide better predictions for new (unknown) values.

If you deselect this option, the model can accept only values that were present in the training data.

10. Connect a labeled dataset, and one of the training modules:

- If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
- If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#) module. With this option, the algorithm iterates over multiple combinations of the settings you provided and determines the combination of values that produces the best model.

NOTE

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

11. Run the experiment.

Results

After training is complete:

- To use the model for scoring, connect it to [Score Model](#), to predict values for new input examples.

Examples

For examples of how decision forests are used in machine learning, see the [Azure AI Gallery](#):

- [Compare Multiclass Classifiers sample](#): Uses several algorithms and discusses their pros and cons.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Related research

For more information about the training process with the **Replicate** option, see:

- [Decision forests for computer vision and medical image analysis. Criminisi and Shotton. Springer 2013](#)

Usage tips

If you have limited data or want to minimize the time spent training the model, try these recommendations:

Limited training set

If the training set contains a limited number of instances:

- Create the decision jungle using a large number of decision DAGs (for example, more than 20)
- Use the **Bagging** option for resampling.
- Specify a large number of optimization steps per DAG layer (for example, more than 10,000).

Limited training time

If the training set contains a large number of instances and training time is limited:

- Create the decision jungle that uses a smaller number of decision DAGs (for example, 5-10).
- Use the **Replicate** option for resampling.
- Specify a smaller number of optimization steps per DAG layer (for example, less than 2000).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Resampling method	Any	ResamplingMethod	Bagging	Choose a resampling method
Number of decision DAGs	>=1	Integer	8	Specify the number of decision graphs that can be created in the ensemble
Maximum depth of the decision DAGs	>=1	Integer	32	Specify the maximum depth of the decision graphs to create in the ensemble
Maximum width of the decision DAGs	>=8	Integer	128	Specify the maximum width of the decision graphs to create in the ensemble
Number of optimization steps per decision DAG layer	>=1000	Integer	2048	Specify the number of steps to use to optimize each level of the decision graphs

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Allow unknown values for categorical features	Any	Boolean	True	Indicate whether unknown values of existing categorical features can be mapped to a new, additional feature

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained multiclass classification model

See also

[Two-Class Decision Jungle](#)

[Classification](#)

[A-Z Module List](#)

Multiclass Logistic Regression

3/10/2021 • 6 minutes to read • [Edit Online](#)

Creates a multiclass logistic regression classification model

Category: [Machine Learning](#) / [Initialize Model](#) / [Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Multiclass Logistic Regression** module in Azure Machine Learning Studio (classic), to create a logistic regression model that can be used to predict multiple values.

Classification using logistic regression is a supervised learning method, and therefore requires a labeled dataset. You train the model by providing the model and the labeled dataset as an input to a module such as [Train Model](#) or [Tune Model Hyperparameters](#). The trained model can then be used to predict values for new input examples.

Azure Machine Learning Studio (classic) also provides a [Two-Class Logistic Regression](#) module, which is suited for classification of binary or dichotomous variables.

More about multiclass logistic regression

Logistic regression is a well-known method in statistics that is used to predict the probability of an outcome, and is particularly popular for classification tasks. The algorithm predicts the probability of occurrence of an event by fitting data to a logistic function. For details about this implementation, see the [Technical Notes](#) section.

In multiclass logistic regression, the classifier can be used to predict multiple outcomes.

How to configure a Multiclass Logistic Regression

1. Add the **Multiclass Logistic Regression** module to the experiment.
2. Specify how you want the model to be trained, by setting the **Create trainer mode** option.
 - **Single Parameter:** Use this option if you know how you want to configure the model, and provide a specific set of values as arguments.
 - **Parameter Range:** Use this option if you are not sure of the best parameters, and want to use a parameter sweep.
3. **Optimization tolerance**, specify the threshold value for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model.
4. **L1 regularization weight**, **L2 regularization weight**: Type a value to use for the regularization parameters L1 and L2. A non-zero value is recommended for both.

Regularization is a method for preventing overfitting by penalizing models with extreme coefficient values. 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. L1 can be applied to sparse models, which is useful when working with high-dimensional data. In contrast, L2 regularization is preferable for data that is not sparse. This algorithm supports a linear combination of L1 and L2 regularization values: that is, if $x = L1$ and $y = L2$, $ax + by = c$ defines the linear span of the regularization terms.

Different linear combinations of L1 and L2 terms have been devised for logistic regression models, such as [elastic net regularization](#).

5. **Memory size for L-BFGS:** Specify the amount of memory to use for *L-BFGS* optimization. This parameter indicates the number of past positions and gradients to store for the computation of the next step.

L-BFGS stands for limited memory Broyden-Fletcher-Goldfarb-Shanno, and it is an optimization algorithm that is popular for parameter estimation. This optimization parameter limits the amount of memory that is used to compute the next step and direction. When you specify less memory, training is faster but less accurate.

6. **Random number seed:** Type an integer value to use as the seed for the algorithm if you want the results to be repeatable over runs. Otherwise, a system clock value is used as the seed, which can produce slightly different results in runs of the same experiment.
7. **Allow unknown categorical levels:** Select this option to create an additional "unknown" level in each categorical column. Any values (levels) in the test dataset that are not present in the training dataset are mapped to this "unknown" level.
8. Connect a labeled dataset, and one of the train modules:
 - If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
 - If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#) module. With this option, you can specify multiple values, and the trainer iterates over multiple combinations of the settings to determine the combination of values that produces the best model.

NOTE

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

9. Run the experiment.

Results

After training is complete:

- To see a summary of the model's parameters, together with the feature weights learned from training, right-click the output of the [Train Model](#) module or [Tune Model Hyperparameters](#), and select **Visualize**.

Examples

For examples of how this learning algorithm is used, see the [Azure AI Gallery](#):

- [Iris clustering](#): Compares the results of multiclass logistic regression with K-means clustering.

- [Network intrusion detection](#): Uses binary logistic regression to determine if a case represents an intrusion.
- [Cross Validation for Binary Classifiers](#): Demonstrates the use of logistic regression in a typical experimental workflow, including model evaluation.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Related research

Want to learn more about L1 and L2 regularization? The following article provides a discussion of how L1 and L2 regularization are different and how they affect model fitting, with code samples for logistic regression and neural network models.

- [L1 and L2 Regularization for Machine Learning](#)

For more information on the implementation of this algorithm, see:

- [Scalable Training of L-1 Regularized Log-Linear Models](#), by Andrew and Gao.

Implementation details

Logistic regression requires numeric variables. Therefore, when you try to use categorical columns as a variable, Azure Machine Learning converts the values to an indicator array internally.

For dates and times, a numeric representation is used. For more information about date time values, see [DateTime Structure .NET Framework](#). If you want to handle dates and times differently we suggest that you create a derived column.

Standard logistic regression is binomial and assumes two output classes. Multiclass or multinomial logistic regression assumes three or more output classes.

Binomial logistic regression assumes a *logistic distribution* of the data, where the probability that an example belongs to class 1 is the formula:

$$p(x; \beta_0, \dots, \beta_{D-1})$$

Where:

- x is a D-dimensional vector containing the values of all the features of the instance.
- p is the logistic distribution function.
- $\beta_0, \dots, \beta_{D-1}$ are the unknown parameters of the logistic distribution.

The algorithm tries to find the optimal values for $\beta_0, \dots, \beta_{D-1}$ by maximizing the log probability of the parameters given the inputs. Maximization is performed by using a popular method for parameter estimation, called [Limited Memory BFGS](#).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Optimization tolerance	<code>>=double.Epsilon</code>	Float	0.000001	Specify a tolerance value for the L-BFGS optimizer

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
L1 regularization weight	$>=0.0$	Float	1.0	Specify the L1 regularization weight. Use a non-zero value to avoid overfitting.
L2 regularization weight	$>=0.0$	Float	1.0	Specify the L2 regularization weight. Use a non-zero value to avoid overfitting.
Memory size for L-BFGS	$>=1$	Integer	20	Specify the amount of memory (in MB) to use for the L-BFGS optimizer. When less memory is used, training is faster, but less accurate.
Random number seed	Any	Integer		Type a value to seed the random number generator used by the model. Leave it blank for the default.
Allow unknown categorical levels	Any	Boolean	True	Indicate whether an additional level should be created for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained classification model

See also

[Classification](#)

[Two-Class Logistic Regression](#)

[A-Z Module List](#)

Multiclass Neural Network

3/10/2021 • 9 minutes to read • [Edit Online](#)

Creates a multiclass classification model using a neural network algorithm

Category: [Machine Learning / Initialize Model / Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Multiclass Neural Network** module in Azure Machine Learning Studio (classic), to create a neural network model that can be used to predict a target that has multiple values.

For example, neural networks of this kind might be used in complex computer vision tasks, such as digit or letter recognition, document classification, and pattern recognition.

Classification using neural networks is a supervised learning method, and therefore requires a *tagged dataset* that includes a label column.

You can train the model by providing the model and the tagged dataset as an input to [Train Model](#) or to [Tune Model Hyperparameters](#). The trained model can then be used to predict values for the new input examples.

More about neural networks

A neural network is a set of interconnected layers. The inputs are the first layer, and are connected to an output layer by an acyclic graph comprised of weighted edges and nodes.

Between the input and output layers you can insert multiple hidden layers. Most predictive tasks can be accomplished easily with only one or a few hidden layers. However, recent research has shown that deep neural networks (DNN) with many layers can be very effective in complex tasks such as image or speech recognition. The successive layers are used to model increasing levels of semantic depth.

The relationship between inputs and outputs is learned from training the neural network on the input data. The direction of the graph proceeds from the inputs through the hidden layer and to the output layer. All nodes in a layer are connected by the weighted edges to nodes in the next layer.

To compute the output of the network for a particular input, a value is calculated at each node in the hidden layers and in the output layer. The value is set by calculating the weighted sum of the values of the nodes from the previous layer. An activation function is then applied to that weighted sum.

How to configure Multiclass Neural Network

1. Add the **MultiClass Neural Network** module to your experiment in Studio (classic). You can find this module under **Machine Learning**, **Initialize**, in the **Classification** category.
2. **Create trainer mode:** Use this option to specify how you want the model to be trained:
 - **Single Parameter:** Choose this option if you already know how you want to configure the model.

- **Parameter Range:** Choose this option if you are not sure of the best parameters, and want to use a parameter sweep. You then specify a range of values and use the [Tune Model Hyperparameters](#) module to iterate over the combinations and find the optimal configuration.

3. **Hidden layer specification:** Select the type of network architecture to create.

- **Fully-connected case:** Select this option to create a model using the default neural network architecture. For multiclass neural network models, the defaults are as follows:
 - One hidden layer
 - The output layer is fully connected to the hidden layer.
 - The hidden layer is fully connected to the input layer.
 - The number of nodes in the input layer is determined by the number of features in the training data.
 - The number of nodes in the hidden layer can be set by the user. The default is 100.
 - The number of nodes in the output layer depends on the number of classes.
- **Custom definition script.** Choose this option to create a custom neural network architecture, using the Net# language. You can define the number of hidden layers, their connections, and advanced options such as specifying the mappings between layers. For an introduction to Net#, see [More About Net#](#) later in this topic.

4. **Neural network definition:** If you selected the custom architecture option, use the text box to type or paste in statements written in the Net# language. For additional script examples, see [Guide to the Net# Neural Networks Specification Language](#).

5. **Number of hidden nodes:** This option lets you customize the number of hidden nodes in the default architecture. Type the number of hidden nodes. The default is one hidden layer with 100 nodes.
6. **The learning rate:** Define the size of the step taken at each iteration, before correction. A larger value for learning rate can cause the model to converge faster, but it can overshoot local minima.
7. **Number of learning iterations:** Specify the maximum number of times the algorithm should process the training cases.
8. **The initial learning weights diameter:** Specify the node weights at the start of the learning process.
9. **The momentum:** Specify a weight to apply during learning to nodes from previous iterations.

10. **The type of normalizer:** Select the method to use for feature normalization. The following normalization methods are supported:

- **Binning normalizer:** The binning normalizer creates bins of equal size, and then normalizes every value in each bin, by dividing by the total number of bins.
- **Gaussian normalizer:** The Gaussian normalizer rescales the values of each feature to have mean 0 and variance 1. This is done by computing the mean and the variance of each feature. For each instance, the mean value is subtracted, and the result divided by the square root of the variance (the standard deviation).
- **Min-max normalizer:** The min-max normalizer linearly rescales every feature to the [0,1] interval.
Rescaling to the [0,1] interval is done by shifting the values of each feature so that the minimal value is 0, and then dividing by the new maximal value (which is the difference between the original maximal and minimal values).
- **Do not normalize:** No normalization is performed.

11. **Shuffle examples:** Select this option to shuffle cases between iterations.

If you deselect this option, cases are processed in exactly the same order each time you run the experiment.

12. **Random number seed:** Type a value to use as the seed, if you want to ensure repeatability across runs of the same experiment.

13. **Allow unknown categorical levels:** Select this option to create a grouping for unknown values in the training and validation sets. The model might be less precise on known values but provide better predictions for new (unknown) values.

If you deselect this option, the model can accept only the values contained in the training data.

14. Connect a training dataset and one of the [training modules](#):

- If you set **Create trainer mode** to **Single Parameter**, use [Train Model](#).
- If you set **Create trainer mode** to **Parameter Range**, use [Tune Model Hyperparameters](#).

NOTE

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

Results

After training is complete:

- To see a summary of the model's parameters, together with the feature weights learned from training, and other parameters of the neural network, right-click the output of [Train Model](#) or [Tune Model Hyperparameters](#), and select **Visualize**.
- To save a snapshot of the trained model, right-click the **Trained model** output and select **Save As Trained Model**. This model is not updated on successive runs of the same experiment.
- To perform cross-validation against a labeled data set, connect the untrained model to [Cross-Validate Model](#).

Examples

For examples of how this learning algorithm is used, see these sample experiments in the [Azure AI Gallery](#). The experiments are related and described in a single document that progresses from basic to advanced configurations:

- [Deep Neural networks example \(part A\)](#)
- [Deep Neural networks example \(part B\)](#)
- [Deep Neural networks example \(part C\)](#)
- [Deep Neural networks example \(part D\)](#)

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Customizing the neural network using script

In Azure Machine Learning Studio (classic), you can customize the architecture of a neural network model by using the Net# language. Customizations supported by the Net# language include:

- Specifying the number of hidden layers and the number of nodes in each layer
- Specifying mappings between layers
- Defining convolutions and weight-sharing bundles
- Choosing the activation function

A neural network model is defined by the structure of its graph, which includes these attributes:

- The number of hidden layers
- The number of nodes in each hidden layer
- How the layers are connected
- Which activation function is used
- Weights on the graph edges

IMPORTANT

The overall structure of the graph, as well as the activation function, can be specified by the user. However, the weights on the edges cannot be specified, and must be learned when training the neural network on the input data.

In general, the network has these defaults:

- The first layer is always the input layer.
- The last layer is always the output layer.
- The number of nodes in the output layer should be equal to the number of classes.

You can define any number of intermediate layers: these are sometimes called hidden layers, because they are contained within the model, and they are not directly exposed as endpoints.

The Net# reference guide explains the syntax and provides sample network definitions. It explains how you can use Net# to add hidden layers and define the way that the different layers interact with each other.

For example, the following script uses the `auto` keyword, which sets the number of features automatically for input and output layers, and uses the default values for the hidden layer.

```
input Data auto;
hidden Hidden auto from Data all;
output Result auto from Hidden all;
```

For additional script examples, see [Guide to the Net# Neural Networks Specification Language](#).

TIP

Neural networks can be computationally expensive, due to a number of hyperparameters and the introduction of custom network topologies. Although in many cases neural networks produce better results than other algorithms, obtaining such results may involve fair amount of sweeping (iterations) over hyperparameters.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Hidden layer specification	List	Neural Network Topology	Fully-connected case	Specify the architecture of the hidden layer or layers
The initial learning weights diameter	$>=\text{double.Epsilon}$	Float	0.1	Specify the node weights at the start of the learning process
The learning rate	[double.Epsilon;1.0]	Float	0.1	Specify the size of each step in the learning process
The momentum	[0.0;1.0]	Float	0.0	Specify a weight to apply during learning to nodes from previous iterations
Neural network definition	Any	StreamReader		When you select Custom definition script , type a valid script expression on each line to define the layers, nodes, and behavior of a custom neural network
The type of normalizer	List	Normalization Method	Minimum-maximum normalizer	Select the type of normalization to apply to learning examples
Number of learning iterations	$>=1$	Integer	100	Specify the number of iterations while learning
Shuffle examples	Any	Boolean	True	Select this option to change the order of instances between learning iterations
Random number seed	Any	Integer		Specify a numeric seed to use for random number generation. Leave blank to use the default seed.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Allow unknown categorical levels	Any	Boolean	True	Indicate whether an additional level should be created for unknown categories. If the test dataset contains categories that are not present in the training dataset, they are mapped to this unknown level.

Output

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained multiclass classification model

See also

[Classification](#)

[Two-Class Neural Network](#)

[Neural Network Regression](#)

[A-Z Module List](#)

One-vs-All Multiclass

3/10/2021 • 3 minutes to read • [Edit Online](#)

Creates a multiclass classification model from an ensemble of binary classification models

Category: [Machine Learning / Initialize Model / Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **One-Vs-All Multiclass** module in Azure Machine Learning Studio (classic), to create a classification model that can predict multiple classes, using the "one vs. all" approach.

This module is useful for creating models that predict three or more possible outcomes, when the outcome depends on continuous or categorical predictor variables. This method also lets you use binary classification methods for issues that require multiple output classes.

More about one-vs.all models

While some classification algorithms permit the use of more than two classes by design, others restrict the possible outcomes to one of two values (a binary, or two-class model). However, even binary classification algorithms can be adapted for multi-class classification tasks using a variety of strategies.

This module implements the *one vs. all method*, in which a binary model is created for each of the multiple output classes. Each of these binary models for the individual classes is assessed against its complement (all other classes in the model) as though it were a binary classification issue. Prediction is then performed by running these binary classifiers, and choosing the prediction with the highest confidence score.

In essence, an ensemble of individual models is created and the results are then merged, to create a single model that predicts all classes. Thus, any binary classifier can be used as the basis for a one-vs-all model.

For example, let's say you configure a [Two-Class Support Vector Machine](#) model and provide that as input to the **One-Vs-All Multiclass** module. The module would create two-class support vector machine models for all members of the output class and then apply the one-vs-all method to combine the results for all classes.

How to Configure the One-vs-All Classifier

This module creates an ensemble of binary classification models to analyze multiple classes. Therefore, to use this module, you need to configure and train a **binary classification** model first.

You then connect the binary model to **One-Vs-All Multiclass** module, and train the ensemble of models by using [Train Model](#) with a labeled training dataset.

When you combine the models, even though the training dataset might have multiple class values, the **One-Vs-All Multiclass** creates multiple binary classification models, optimizes the algorithm for each class, and then merges the models.

1. Add the **One-Vs-All Multiclass** to your experiment in Studio (classic). You can find this module under

Machine Learning - Initialize, in the **Classification** category.

The **One-Vs-All Multiclass** classifier has no configurable parameters of its own. Any customizations must be done in the binary classification model that is provided as input.

2. Add a binary classification model to the experiment, and configure that model. For example, you might use a [Two-Class Support Vector Machine](#) or [Two-Class Boosted Decision Tree](#).

If you need help choosing the right algorithm, see these resources:

- [Machine learning algorithm cheat sheet for Azure ML](#)
- [How to choose algorithms for Azure Machine Learning Studio \(classic\)](#)

3. Add the [Train Model](#) module to your experiment, and connect the untrained classifier that is the output of **One-Vs-All Multiclass**.
4. On the other input of [Train Model](#), connect a labeled training data set that has multiple class values.
5. Run the experiment, or select **Train Model** and click **Run Selected**.

Results

After training is complete, you can use the model to make multiclass predictions.

Alternatively, you can pass the untrained classifier to [Cross-Validate Model](#) for cross-validation against a labeled validation data set.

Examples

For examples of how this learning algorithm is used, see the [Azure AI Gallery](#):

- [News Categorization](#): This sample uses **One-Vs-All Multiclass** with a [Two-Class Decision Forest](#) model.
- [Compare Multiclass Classifier sample](#): Binary classifiers are used for each digit and the results are combined.

Expected inputs

NAME	TYPE	DESCRIPTION
Untrained binary classification model	ILearner interface	An untrained binary classification model

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained multiclass classification

Exceptions

EXCEPTION	DESCRIPTION
Error 0013	An exception occurs if the learner that was passed to the module is the wrong type.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Classification](#)

Two-Class Averaged Perceptron

3/10/2021 • 4 minutes to read • [Edit Online](#)

Creates an averaged perceptron binary classification model

Category: [Machine Learning / Initialize Model / Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Two-Class Averaged Perceptron** module in Azure Machine Learning Studio (classic), to create a machine learning model based on the averaged perceptron algorithm.

This classification algorithm is a supervised learning method, and requires a *tagged dataset*, which includes a label column. You can train the model by providing the model and the tagged dataset as an input to [Train Model](#) or [Tune Model Hyperparameters](#). The trained model can then be used to predict values for the new input examples.

More about averaged perceptron models

The *averaged perceptron method* is an early and very simple version of a neural network. In this approach, inputs are classified into several possible outputs based on a linear function, and then combined with a set of weights that are derived from the feature vector—hence the name "perceptron."

The simpler perceptron models are suited to learning linearly separable patterns, whereas neural networks (especially deep neural networks) can model more complex class boundaries. However, perceptrons are faster, and because they process cases serially, perceptrons can be used with continuous training.

How to configure Two-Class Averaged Perceptron

1. Add the **Two-Class Averaged Perceptron** module to your experiment in Studio (classic).
2. Specify how you want the model to be trained, by setting the **Create trainer mode** option.
 - **Single Parameter:** If you know how you want to configure the model, provide a specific set of values as arguments.
 - **Parameter Range:** If you are not sure of the best parameters, find the optimal parameters by specifying multiple values and using the [Tune Model Hyperparameters](#) module to find the optimal configuration. The trainer iterates over multiple combinations of the settings you provided and determines the combination of values that produces the best model.
3. For **Learning rate**, specify a value for the *learning rate*. The learning rate values controls the size of the step that is used in stochastic gradient descent each time the model is tested and corrected.

By making the rate smaller, you test the model more often, with the risk that you might get stuck in a local plateau. By making the step larger, you can converge faster, at the risk of overshooting the true minima.

4. For **Maximum number of iterations**, type the number of times you want the algorithm to examine the training data.

Stopping early often provides better generalization. Increasing the number of iterations improves fitting, at the risk of overfitting.

5. For **Random number seed**, optionally type an integer value to use as the seed. Using a seed is recommended if you want to ensure reproducibility of the experiment across runs.

6. Select the **Allow unknown categorical levels** option to create a group for unknown values in the training and validation sets. The model might be less precise for known values, but it can provide better predictions for new (unknown) values.

If you deselect this option, the model can accept only the values that are contained in the training data.

7. Connect a training dataset, and one of the [training modules](#):

- If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
- If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#) module.

NOTE

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

Results

After training is complete:

- To see a summary of the model's parameters, together with the feature weights learned from training, right-click the output of [Train Model](#) or [Tune Model Hyperparameters](#).

Examples

For examples of how this learning algorithm is used, see the [Azure AI Gallery](#):

- [Cross Validation for Binary Classifiers sample](#): Compares multiple classification models.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Usage tips

For this model type, it is a best practice to normalize datasets before using them to train the classifier. For normalization options, see [Normalize Data](#).

The averaged perceptron model is an early and simplified version of neural networks. As such, it works well on simple data sets when your goal is speed over accuracy. However, if you are not getting the desired results, try one of these models:

- [Two-Class Neural Network](#) or [Multiclass Neural Network](#)
- [Two-Class Logistic Regression](#) or [Multiclass Logistic Regression](#)

- [Two-Class Boosted Decision Tree](#)

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Learning rate	$>= \text{double.Epsilon}$	Float	1.0	The initial learning rate for the Stochastic Gradient Descent optimizer.
Maximum number of iterations	$>= 1$	Integer	10	The number of Stochastic Gradient Descent iterations to be performed over the training dataset.
Random number seed	Any	Integer		The seed for the random number generator used by the model. Leave it blank for the default.
Allow unknown categorical levels	Any	Boolean	True	If True, creates an additional level for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

Output

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained binary classification model that can be connected to the One-vs-All Multiclass , Train Model , or Cross-Validate Model modules.

See also

[Classification](#)

[A-Z Module List](#)

Two-Class Bayes Point Machine

3/10/2021 • 3 minutes to read • [Edit Online](#)

Creates a Bayes point machine binary classification model

Category: [Machine Learning / Initialize Model / Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Two-Class Bayes Point Machine** module in Azure Machine Learning Studio (classic), to create an untrained binary classification model.

The algorithm in this module uses a Bayesian approach to linear classification called the "Bayes Point Machine". This algorithm efficiently approximates the theoretically optimal Bayesian average of linear classifiers (in terms of generalization performance) by choosing one "average" classifier, the Bayes Point. Because the Bayes Point Machine is a Bayesian classification model, it is not prone to overfitting to the training data.

For more information, see Chris Bishop's post on the Microsoft Machine Learning blog: [Embracing Uncertainty - Probabilistic Inference](#).

How to configure Two-Class Bayes Point Machine

1. In Azure Machine Learning Studio (classic), add the **Two-Class Bayes Point Machine** module to your experiment. You can find the module under **Machine Learning**, **Initialize Model**, **Classification**.
2. For **Number of training iterations**, type a number to specify how often the message-passing algorithm iterates over the training data. Typically, the number of iterations should be set to a value in the range 5 – 100.

The higher the number of training iterations, the more accurate the predictions; however, training will be slower.

For most datasets, the default setting of 30 training iterations is sufficient for the algorithm to make accurate predictions. Sometimes accurate predictions can be made by using fewer iterations. For datasets with highly correlated features, you might benefit from more training iterations.

3. Select the option, **Include bias**, if you want a constant feature or bias to be added to each instance in training and prediction.

Including a bias is necessary when the data does not already contain a constant feature.

4. Select the option, **Allow unknown values in categorical features**, to create a group for unknown values.

If you deselect this option, the model can accept only the values that are contained in the training data.

If you select this option and allow unknown values, the model might be less precise for known values, but

it can provide better predictions for new (unknown) values.

5. Add an instance of the [Train Model](#) module, and your training data.
6. Connect the training data and the output of the [Two-Class Bayes Point Machine](#) module to the [Train Model](#) module, and choose the label column.
7. Run the experiment.

Results

After training is complete, right-click the output of the [Train Model](#) module to view the results:

- To see a summary of the model's parameters, together with the feature weights learned from training, select [Visualize](#).
- To save the model for later use, right-click the output of [Train Model](#), and select [Save as Trained Model](#).
- To make predictions, use the trained model as an input to the [Score Model](#) module.

The untrained model can also be passed to [Cross-Validate Model](#) for cross-validation against a labeled data set.

Examples

To see how the Two-Class Bayes Point Machine is used in machine learning, see these sample experiments in the [Azure AI Gallery](#):

- [Compare Binary Classifiers](#): This sample demonstrates the use of multiple two-class classifiers.

Technical notes

This section contains implementation details and frequently asked questions about this algorithm.

Details from the original research and underlying theory are available in this paper (PDF): [Bayes Point Machines, by Herbert, Graepel, and Campbell](#)

However, this implementation improves on the original algorithm in several ways:

- It uses the expectation propagation message-passing algorithm. For more information, see [A family of algorithms for approximate Bayesian inference](#).
- A parameter sweep is not required.
- This method does not require data to be normalized.

These improvements make the Bayes Point Machine classification model more robust and easier-to-use, and you can bypass the time-consuming step of parameter tuning.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Number of training iterations	>=1	Integer	30	Specify the number of iterations to use when training

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Include bias	Any	Boolean	True	Indicate whether a constant feature or bias should be added to each instance
Allow unknown values in categorical features	Any	Boolean	True	If True, creates an additional level for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

Output

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained binary classification model

See also

[Classification A-Z Module List](#)

Two-Class Boosted Decision Tree

3/24/2021 • 8 minutes to read • [Edit Online](#)

Creates a binary classifier using a boosted decision tree algorithm

Category: [Machine Learning / Initialize Model / Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Two-Class Boosted Decision Tree** module in Azure Machine Learning Studio (classic), to create a machine learning model that is based on the boosted decision trees algorithm.

A boosted decision tree is an ensemble learning method in which the second tree corrects for the errors of the first tree, the third tree corrects for the errors of the first and second trees, and so forth. Predictions are based on the entire ensemble of trees together that makes the prediction. For further technical details, see the [Research](#) section of this article.

Generally, when properly configured, boosted decision trees are the easiest methods with which to get top performance on a wide variety of machine learning tasks. However, they are also one of the more memory-intensive learners, and the current implementation holds everything in memory. Therefore, a boosted decision tree model might not be able to process the very large datasets that some linear learners can handle.

For more information about how to choose an algorithm, see these resources:

- [Machine learning algorithm cheat sheet for Microsoft Azure Machine Learning Studio \(classic\)](#)
- [How to choose Azure Machine Learning algorithms for clustering, classification, or regression](#)

How to configure Two-Class Boosted Decision Tree

This module creates an untrained classification model. Because classification is a supervised learning method, to train the model, you need a *tagged dataset* that includes a label column with a value for all rows.

You can train this type of model by using either the [Train Model](#) or [Tune Model Hyperparameters](#) modules.

1. In Azure Machine Learning Studio (classic), add the **Boosted Decision Tree** module to your experiment.
2. Specify how you want the model to be trained, by setting the **Create trainer mode** option.
 - **Single Parameter:** If you know how you want to configure the model, you can provide a specific set of values as arguments.
 - **Parameter Range:** If you are not sure of the best parameters, you can find the optimal parameters by using the [Tune Model Hyperparameters](#) module. You provide some range of values, and the trainer iterates over multiple combinations of the settings to determine the combination of values that produces the best result.

3. For **Maximum number of leaves per tree**, indicate the maximum number of terminal nodes (leaves) that can be created in any tree.

By increasing this value, you potentially increase the size of the tree and get better precision, at the risk of overfitting and longer training time.

4. For **Minimum number of samples per leaf node**, indicate the number of cases required to create any terminal node (leaf) in a tree.

By increasing this value, you increase the threshold for creating new rules. For example, with the default value of 1, even a single case can cause a new rule to be created. If you increase the value to 5, the training data would have to contain at least 5 cases that meet the same conditions.

5. For **Learning rate**, type a number between 0 and 1 that defines the step size while learning.

The learning rate 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 on the best solution.

6. For **Number of trees constructed**, indicate the total number of decision trees to create in the ensemble. By creating more decision trees, you can potentially get better coverage, but training time will increase.

This value also controls the number of trees displayed when visualizing the trained model. If you want to see or print a single tree, set the value to 1. However, when you do so, only one tree is produced (the tree with the initial set of parameters) and no further iterations are performed.

7. For **Random number seed**, optionally type a non-negative integer to use as the random seed value. Specifying a seed ensures reproducibility across runs that have the same data and parameters.

The random seed is set by default to 0, which means the initial seed value is obtained from the system clock. Successive runs using a random seed can have different results.

8. Select **Allow unknown categorical levels** option to create a group for unknown values in the training and validation sets.

If you deselect this option, the model can accept only the values that are contained in the training data.

If you allow unknown values, the model might be less precise for known values, but likely can provide better predictions for new (unknown) values.

9. Train the model.

- If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the [Train Model](#) module.
- If you set **Create trainer mode** to **Parameter Range**, connect a tagged dataset and train the model by using [Tune Model Hyperparameters](#).

NOTE

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values, and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

Results

After training is complete:

- To see the tree that was created on each iteration, right-click **Train Model** module and select **Trained model** to visualize. If you use **Tune Model Hyperparameters**, right click the module and select **Trained best model** to visualize the best model.

Click each tree to drill down into the splits and see the rules for each node.

- To use the model for scoring, connect it to **Score Model**, to predict values for new input examples.

Examples

For examples of how boosted decision trees are used in machine learning, see the [Azure AI Gallery](#):

- [Direct marketing](#): Uses the **Two-Class Boosted Decision Tree** algorithm to predict customer appetency.
- [Flight delay prediction](#): This sample uses the **Two-Class Boosted Decision Tree** algorithm to determine whether a flight is likely to be delayed.
- [Credit card risk](#): This sample uses the **Two-Class Boosted Decision Tree** algorithm to predict risk.

Technical notes

This section contains implementation details and frequently asked questions.

Usage tips

- To train a boosted decision tree model, you must provide multiple data instances. An error is generated during the training process if the dataset contains too few rows.
- If your data has missing values, you must add indicators for the features.
- In general, boosted decision trees yield better results when features are somewhat related. If features have a large degree of entropy (that is, they are not related), they share little or no mutual information, and ordering them in a tree does not yield a lot of predictive significance. If this is not the case, you might try a random forests model.

Boosting also works well when you have many more examples than features because the model is prone to overfitting.

- Do not normalize the dataset. Because the treatment of features is a simple, non-parametric, less-than or greater-than comparison, normalization or any form of non-monotonic transformation function might have little effect.
- Features are discretized and binned prior to training, so only a relatively small set of threshold candidates are considered, even for continuous features.

Implementation details

For detailed information about the boosted decision tree algorithm, see [Greedy Function Approximation: A Gradient Boosting Machines](#).

The boosted decision tree algorithm in Azure Machine Learning uses the following boosting method:

1. Start with an empty ensemble of weak learners.
2. For each training example, get the current output of the ensemble. This is the sum of the outputs of all weak learners in the ensemble.
3. Calculate the gradient of the loss function for each example.

This depends on whether the task is a binary classification problem or a regression problem.

- In a binary classification model, the log-loss is used, much like in logistic regression.
- In a [regression](#) model, the squared loss is used, and the gradient is the current output, minus the target).

4. Use the examples to fit a **weak learner**, using the gradient just defined as the target function.
5. Add that weak learner to the ensemble with a strength indicated by the learning rate, and if desired, go to Step 2.

In this implementation, the weak learners are the least-squares regression trees, based on the gradients calculated in Step 3. The trees are subject to the following restrictions:

- They are trained up to a maximum number of leaves.
 - Each leaf has a minimum number of examples to guard against overfitting.
 - Each decision node is a single feature that is compared against some threshold. If that feature is less than or equal to the threshold, it goes down one path, and if it is greater than the threshold, it goes down the other path.
 - Each leaf node is a constant value.
6. The tree-building algorithm greedily selects the feature and threshold for which a split minimizes the squared loss with regard to the gradient calculated in Step 3. The selection of the split is subject to a minimum number of training examples per leaf.

The algorithm repeatedly splits until it reaches the maximum number of leaves, or until no valid split is available.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Maximum number of leaves per tree	≥ 1	Integer	20	Specify the maximum number of leaves allowed per tree
Minimum number of samples per leaf node	≥ 1	Integer	10	Specify the minimum number of cases required to form a leaf
Learning rate	[double.Epsilon;1.0]	Float	0.2	Specify the initial learning rate
Number of trees constructed	≥ 1	Integer	100	Specify the maximum number of trees that can be created during training
Random number seed	Any	Integer		Type a value to seed the random number generator that is used by the model. Leave it blank for the default.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Allow unknown categorical levels	Any	Boolean	True	If True, an additional level is created for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

Output

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained binary classification model

See also

[Classification](#)

[Boosted Decision Tree Regression](#)

[A-Z Module List](#)

Two-Class Decision Forest

3/24/2021 • 8 minutes to read • [Edit Online](#)

Creates a two-class classification model using the decision forest algorithm

Category: [Machine Learning / Initialize Model / Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Two-Class Decision Forest** module in Azure Machine Learning Studio (classic), to create a machine learning model based on the decision forests algorithm.

Decision forests are fast, supervised ensemble models. This module is a good choice if you want to predict a target with a maximum of two outcomes. If you are not sure how to configure a decision tree model for the best results, we recommend that you use the [Tune Model Hyperparameters](#) module to train and test multiple models. Tuning iterates over multiple possibilities and finds the optimal solution for you.

Understanding decision forests

This decision forest algorithm is an ensemble learning method intended for classification tasks. Ensemble methods are based on the general principle that rather than relying on a single model, you can get better results and a more generalized model by creating multiple related models and combining them in some way. Generally, ensemble models provide better coverage and accuracy than single decision trees.

There are many ways to create individual models and combine them in an ensemble. This particular implementation of a decision forest works by building multiple decision trees and then **voting** on the most popular output class. Voting is one of the better-known methods for generating results in an ensemble model.

- Many individual classification trees are created, using the entire dataset, but different (usually randomized) starting points. This differs from the random forest approach, in which the individual decision trees might only use some randomized portion of the data or features.
- Each tree in the decision forest tree outputs a non-normalized frequency histogram of labels.
- The aggregation process sums these histograms and normalizes the result to get the “probabilities” for each label.
- The trees that have high prediction confidence will have a greater weight in the final decision of the ensemble.

Decision trees in general have many advantages for classification tasks:

- They can capture non-linear decision boundaries.
- You can train and predict on lots of data, as they are efficient in computation and memory usage.
- Feature selection is integrated in the training and classification processes.
- Trees can accommodate noisy data and many features.
- They are non-parametric models, meaning they can handle data with varied distributions.

However, simple decision trees can overfit on data, and are less generalizable than tree ensembles.

For more information, see [Decision Forests](#), or the other papers listed in the [Technical notes](#) section.

How to configure Two-Class Decision Forest

1. Add the **Two-Class Decision Forest** module to your experiment in Azure Machine Learning Studio (classic), and open the **Properties** pane of the module.

You can find the module under **Machine Learning**. Expand **Initialize**, and then **Classification**.

2. For **Resampling method**, choose the method used to create the individual trees. You can choose from **Bagging** or **Replicate**.

- **Bagging:** Bagging is also called *bootstrap aggregating*. In this method, each tree is grown on a new sample, created by randomly sampling the original dataset with replacement until you have a dataset the size of the original.

The outputs of the models are combined by *voting*, which is a form of aggregation. Each tree in a classification decision forest outputs an un-normalised frequency histogram of labels. The aggregation is to sum these histograms and normalise to get the “probabilities” for each label. In this manner, the trees that have high prediction confidence will have a greater weight in the final decision of the ensemble.

For more information, see the Wikipedia entry for [Bootstrap aggregating](#).

- **Replicate:** In replication, each tree is trained on exactly the same input data. The determination of which split predicate is used for each tree node remains random and the trees will be diverse.

For more information about the training process with the **Replicate** option, see the papers listed in the [Technical Notes](#) section.

3. Specify how you want the model to be trained, by setting the **Create trainer mode** option.

- **Single Parameter:** If you know how you want to configure the model, you can provide a specific set of values as arguments.
- **Parameter Range:** If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using the [Tune Model Hyperparameters](#) module to find the optimal configuration. The trainer iterates over multiple combinations of the settings you provided and determines the combination of values that produces the best model.

4. For **Number of decision trees**, type the maximum number of decision trees that can be created in the ensemble. By creating more decision trees, you can potentially get better coverage, but training time increases.

NOTE

This value also controls the number of trees displayed when visualizing the trained model. If you want to see or print a single tree, you can set the value to 1. However, only one tree can be produced (the tree with the initial set of parameters) and no further iterations are performed.

5. For **Maximum depth of the decision trees**, type a number to limit the maximum depth of any decision tree. Increasing the depth of the tree might increase precision, at the risk of some overfitting and increased training time.
6. For **Number of random splits per node**, type the number of splits to use when building each node of the tree. A *split* means that features in each level of the tree (node) are randomly divided.

7. For **Minimum number of samples per leaf node**, indicate the minimum number of cases that are required to create any terminal node (leaf) in a tree.

By increasing this value, you increase the threshold for creating new rules. For example, with the default value of 1, even a single case can cause a new rule to be created. If you increase the value to 5, the training data would have to contain at least 5 cases that meet the same conditions.

8. Select the **Allow unknown values for categorical features** option to create a group for unknown values in the training or validation sets. The model might be less precise for known values, but it can provide better predictions for new (unknown) values.

If you deselect this option, the model can accept only the values that are contained in the training data.

9. Attach a labeled dataset, and one of the [training modules](#):

- If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
- If you set **Create trainer mode** to **Parameter Range**, use [Tune Model Hyperparameters](#).

NOTE

If you pass a parameter range to [Train Model](#), only the first value in the parameter range list is used.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values, and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value is used throughout the sweep, even if other parameters change across a range of values.

Results

After training is complete:

- To see the tree that was created on each iteration, right-click [Train Model](#) module and select **Trained model** to visualize. If you use [Tune Model Hyperparameters](#), right click the module and select **Trained best model** to visualize the best model.

Click each tree to drill down into the splits and see the rules for each node.

- To save a snapshot of the model, right-click the **Trained Model** output, and select **Save Model**. The saved model is not updated on successive runs of the experiment.
- To use the model for scoring, add the **Score Model** module to an experiment.

Examples

For examples of how decision forests are used in machine learning, see the sample experiments in the [Azure AI Gallery](#):

- [News categorization](#): Compares a multiclass classifier to a model built using the **Two-Class Decision Forest** algorithm with the [One-vs-All Multiclass](#).
- [Predictive maintenance](#): An extended walkthrough that uses the **Two-Class Decision Forest** algorithm to predict if an asset will fail within certain time frame.

Technical notes

This section contains additional implementation details, research, and frequently asked questions.

Usage tips

If you have limited data, or if you want to minimize the time spent training the model, try these settings:

Limited training set

If the training set contains a limited number of instances:

- Create the decision forest by using a large number of decision trees (for example, more than 20).
- Use the **Bagging** option for resampling.
- Specify a large number of random splits per node (for example, more than 1,000).

Limited training time

If the training set contains a large number of instances and training time is limited:

- Create the decision forest by using fewer decision trees (for example, 5-10).
- Use the **Replicate** option for resampling.
- Specify a smaller number of random splits per node (for example, fewer than 100).

Implementation details

This article by Microsoft Research provides useful information about ensemble methods that use decision trees.

[From Stumps to Trees to Forests](#).

For more information about the training process with the **Replicate** option, see [Decision Forests for Computer Vision and Medical Image Analysis](#). Criminisi and J. Shotton. Springer 2013.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Resampling method	Any	ResamplingMethod	Bagging	Choose a resampling method
Number of decision trees	>=1	Integer	8	Specify the number of decision trees to create in the ensemble
Maximum depth of the decision trees	>=1	Integer	32	Specify the maximum depth of any decision tree that can be created
Number of random splits per node	>=1	Integer	128	Specify the number of splits generated per node, from which the optimal split is selected
Minimum number of samples per leaf node	>=1	Integer	1	Specify the minimum number of training samples that are required to produce a leaf node

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Allow unknown values for categorical features	Any	Boolean	True	Indicate whether unknown values of existing categorical features can be mapped to a new, additional feature

Output

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained binary classification model

See also

[Classification](#)
[Decision Forest Regression](#)
[Multiclass Decision Forest](#)
[A-Z Module List](#)

Two-Class Decision Jungle

3/24/2021 • 5 minutes to read • [Edit Online](#)

Creates a two-class classification model using the decision jungle algorithm

Category: [Machine Learning](#) / [Initialize Model](#) / [Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Two-Class Decision Jungle** module in Azure Machine Learning Studio (classic), to create a machine learning model that is based on a supervised ensemble learning algorithm called decision jungles.

The **Two-Class Decision Jungle** module returns an untrained classifier. You then train this model on a labeled training data set, by using [Train Model](#) or [Tune Model Hyperparameters](#). The trained model can then be used to make predictions.

More about decision jungles

[Decision jungles](#) are a recent extension to [decision forests](#). A decision jungle consists of an ensemble of decision directed acyclic graphs (DAGs).

Decision jungles have the following advantages:

- By allowing tree branches to merge, a decision DAG typically has a lower memory footprint and better generalization performance than a decision tree, albeit at the cost of somewhat longer training time.
- Decision jungles are non-parametric models that can represent non-linear decision boundaries.
- They perform integrated feature selection and classification and are resilient in the presence of noisy features.

TIP

For more information about the research behind this machine learning algorithm, see [Decision Jungles: Compact and Rich Models for Classification](#) (downloadable PDF).

How to configure Two-Class Decision Jungle

1. Add the **Two-Class Decision Jungle** module to your experiment in Studio (classic).
2. For **Resampling method**, choose the method used to create the individual trees. You can choose from **Bagging** or **Replicate**.
 - **Bagging**: Select this option to use bagging, also called bootstrap aggregating.

Each tree in a decision jungle outputs a Gaussian distribution as prediction. The aggregation is to

find a Gaussian whose first two moments match the moments of the mixture of Gaussians given by combining all Gaussians returned by individual trees.

- **Replicate:** In replication, each tree is trained on exactly the same input data. The determination of which split predicate is used for each tree node remains random and the trees will be diverse.

For more information, see [Decision Forests for Computer Vision and Medical Image Analysis. Criminisi and J. Shotton. Springer 2013.](#)

3. Specify how you want the model to be trained, by setting the **Create trainer mode** option.

- **Single Parameter:** If you know how you want to configure the model, you can provide a specific set of values as arguments.
- **Parameter Range:** If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using the [Tune Model Hyperparameters](#) module to find the optimal configuration. The trainer will iterate over multiple combinations of the settings you provided and determine the combination of values that produces the best model.

4. For **Number of decision DAGs**, indicate the maximum number of graphs that can be created in the ensemble.

5. For **Maximum depth of the decision DAGs**, indicate the maximum depth of each graph.

6. For **Maximum width of the decision DAGs**, indicate the maximum width of each graph.

7. In **Number of optimization steps per decision DAG layer**, indicate how many iterations over the data to perform when building each DAG.

8. Select the **Allow unknown values for categorical features** option to create a group for unknown values in testing or validation data.

If you deselect it, the model can accept only the values that are contained in the training data. In the former case, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.

9. Add a tagged dataset to the experiment, and connect one of the [training modules](#).

- If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
- If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#) module.

NOTE

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

Results

After training is complete:

- To use the model for scoring, connect it to [Score Model](#), to predict values for new input examples.

Examples

For examples of how decision jungles are used in machine learning, see the [Azure AI Gallery](#):

- [Compare Binary Classifiers](#): Uses several algorithms and discusses their pros and cons.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Usage tips

If you have limited data, or want to minimize the time spent training the model, try these settings.

Limited training set

If your training set is small:

- Create the decision jungle by using a large number of decision DAGs (for example, more than 20).
- Use the **Bagging** option for resampling.
- Specify a large number of optimization steps per DAG layer (for example, more than 10,000).

Limited training time

If the training set is large but training time is limited:

- Create the decision jungle using a fewer number of decision DAGs (for example, 5-10).
- Use the **Replicate** option for resampling.
- Specify a smaller number of optimization steps per DAG layer (for example, less than 2000).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Resampling method	Any	ResamplingMethod	Bagging	Choose a resampling method
Number of decision DAGs	>=1	Integer	8	Specify the number of decision graphs to build in the ensemble
Maximum depth of the decision DAGs	>=1	Integer	32	Specify the maximum depth of the decision graphs in the ensemble
Maximum width of the decision DAGs	>=8	Integer	128	Specify the maximum width of the decision graphs in the ensemble
Number of optimization steps per decision DAG layer	>=1000	Integer	2048	Specify the number of steps to use to optimize each level of the decision graphs

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Allow unknown values for categorical features	Any	Boolean	True	Indicate whether unknown values of existing categorical features can be mapped to a new, additional feature

Output

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained binary classification model

See also

- [Classification](#)
- [Multiclass Decision Jungle](#)
- [A-Z Module List](#)

Two-Class Locally Deep Support Vector Machine

3/10/2021 • 9 minutes to read • [Edit Online](#)

Creates a binary classification model using the locally deep Support Vector Machine algorithm

Category: [Machine Learning / Initialize Model / Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Two-Class Locally Deep Support Vector Machine** module in Azure Machine Learning Studio (classic), to create a two-class, non-linear support vector machines (SVM) classifier that is optimized for efficient prediction.

Support vector machines (SVMs) are an extremely popular and well-researched class of supervised learning models, which can be used in linear and non-linear classification tasks. Recent research has focused on ways to optimize these models to efficiently scale to larger training sets. In this implementation from Microsoft Research, the kernel function that is used for mapping data points to feature space is specifically designed to reduce the time needed for training while maintaining most of the classification accuracy.

This model is a supervised learning method, and therefore requires a *tagged dataset*, which includes a label column.

After you define the model parameters, train it by providing the model and a tagged dataset as input to [Train Model](#) or [Tune Model Hyperparameters](#). The trained model can then be used to predict values for new inputs.

How to configure Two-Class Locally Deep Support Vector Machine

1. Add the **Two-Class Locally-Deep Support Vector Machine** module to your experiment in Studio (classic).
2. Specify how you want the model to be trained, by setting the **Create trainer mode** option.
 - **Single Parameter:** If you know how you want to configure the model, provide a specific set of values as arguments.
 - **Parameter Range:** If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using the [Tune Model Hyperparameters](#) module to find the optimal configuration. The trainer iterates over multiple combinations of the settings you provided and determines the combination of values that produces the best model.
3. For **Depth of the tree**, specify the maximum depth of the tree that can be created by the local deep kernel learning SVM (LD-SVM) model.

The cost of training increases linearly with tree depth; therefore, choose an appropriate depth, depending on how much time you can afford to spend when building the model.

Training time should roughly double as the depth is increased by one.

Prediction accuracy should increase, reach a peak, and then decrease with increasing depth.

4. For **Lambda W**, specify the weight that should be given to the regularization term.

Regularization restricts large value components in the trained classifier. When the number of samples is insufficient given the number of features, you can use L2 regularization to avoid overfitting. Larger values for **Lambda W** mean that more emphasis is placed on regularizing the classifier weights and less on the training-set classification error.

If the default value (0.1) doesn't work well, you should also try {0.0001, 0.001 and 0.01}.

5. For **Lambda Theta**, specify how much space should be left between a region boundary and the closest data point.

This model works by partitioning the data space and feature space into regions. When **Lambda Theta** is minimized in such a way that region boundaries in the trained model are too close to the training data points, the model might yield a low training error, but a high test error, due to overfitting.

To reduce the number of parameters that need to be validated, a good rule of thumb is to set **Lambda Theta** to one-tenth of the value that is used for **Lambda W**. Larger values mean that more emphasis is put on preventing overfitting than on minimizing classification errors in the training set.

If the default value (0.01) doesn't work well, you should also try {0.0001, 0.001 and 0.1}.

6. For **Lambda Theta Prime**, type a value to control the amount of curvature that is allowed in decision boundaries in the model.

Larger values give the model the flexibility to learn curved decision boundaries, while smaller values might constrain the decision boundaries to more of a stepwise linear pattern.

This parameter works in conjunction with the **Sigma** parameter. To reduce the number of parameters that need to be validated, a good rule of thumb is to set **Lambda Theta Prime** to one-tenth the value of **Lambda W**.

If the default value (0.01) doesn't work well, you should also try {0.0001, 0.001 and 0.1}.

7. For **Sigmoid sharpness**, type a value to use for the scaling parameter σ .

Larger values mean that the **tanh** in local kernel Θ (theta) is saturated, whereas a smaller value implies a more linear operating range for theta. You can find the full optimization formula in the [Technical Notes](#) section.

If the default value (1) does not work well, you can also try {0.1, 0.01, 0.001}.

8. In **Number of iterations**, indicate how many times the algorithm should update the classifier parameters with a random subset of examples.

9. For **Feature normalizer**, choose a method to use in normalizing feature values. The following methods are supported:

- **Binning normalizer:** The binning normalizer creates bins of equal size, and then normalizes every value in each bin to be divided by the total number of bins.
- **Gaussian normalizer:** The Gaussian normalizer rescales the values of each feature to have a mean of 0 and a variance of 1. This is done by computing the mean and the variance of each feature. Then, for each instance, the mean value is subtracted, and the result divided by the square root of the variance (the standard deviation).
- **Min-Max normalizer:** The min-max normalizer linearly rescales every feature to the [0,1]

interval.

Rescaling to the [0,1] interval is done by shifting the values of each feature so that the minimal value is 0, and then dividing by the new maximal value (which is the difference between the original maximal and minimal values).

- **Do not normalize:** No normalization is performed.

10. In **Random number seed**, type a value to use as a seed if you want to ensure reproducibility across runs.

11. Select the **Allow unknown categorical levels** option to create a group for unknown values in the testing or validation sets.

If you deselect it, the model can accept only the values that are contained in the training data. In the former case, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.

12. Connect a tagged dataset and one of the [training modules](#):

- If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
- If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#) module.

NOTE

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

13. Run the experiment.

Results

After training is complete:

- To see a summary of the model's parameters, right-click the output of the [Train Model](#) module or [Tune Model Hyperparameters](#) module, and select **Visualize**.
- To save a snapshot of the trained model, right-click the **Trained model** output and select **Save As Trained Model**. This model is not updated on successive runs of the same experiment.
- To perform cross-validation against a labeled data set, connect the untrained model to [Cross-Validate Model](#).

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Usage tips

This LD-SVM classifier is most useful under the following conditions:

- You have a binary classification issue, or you can reduce your issue to a binary classification task.
- You tried a linear classifier, but it did not perform well.

- You tried a non-linear SVM or other classifier, and got good classification accuracy, but it took too long to train the model.
- You can afford to sacrifice prediction accuracy to reduce training time.

LD-SVM models are a good choice when your data is complicated enough that linear models (such as logistic regression) perform poorly. LD-SVM models are also small enough to be used in mobile devices or other scenarios where complex models (such as neural networks) are too big to be consumed efficiently.

Conversely, this model should not be used if you don't care about model size or if a linear model is required for simplicity or prediction speed. There is also no point to changing to LD-SVM if linear classifiers are already giving good results, or if you can get high classification accuracy by adding small amounts of non-linearity.

Implementation details

The LD-SVM model was developed by Microsoft Research as part of ongoing efforts to speed up non-linear SVM prediction. The work of Gonen and Alpaydin (2008) on the localized multiple kernel learning method was particularly valuable. Use of a local kernel function enables the model to learn arbitrary local feature embeddings, including high-dimensional, sparse, and computationally deep features that introduce non-linearities into the model.

LD-SVM is faster than most other classifiers for several reasons:

- The model learns decision boundaries that are locally linear. Therefore, a test point can be efficiently classified by testing it against its local decision boundary, rather than testing against the entire set of decision boundaries all over feature space.
- The model uses efficient primal-based routines to optimize the space of tree-structured local feature embeddings that scale to large training sets with more than half a million training points.
- The cost of testing a point against its local decision boundary is logarithmic in the number of training points.

As a consequence of these optimizations, training the LD-SVM model is exponentially faster than training traditional SVM models.

Optimization formula

$$\min_{W, \theta, \theta'} P(W, \theta, \theta') = \frac{\lambda_W}{2} \text{Tr}(W^t W) + \frac{\lambda_\theta}{2} \text{Tr}(\theta^t \theta) + \frac{\lambda_{\theta'}}{2} \text{Tr}(\theta'^t \theta') + \sum_{i=1}^N L(y_i, \phi_L^t(x_i) W^t x_i)$$

Research

For more information about the algorithm and underlying research, see [Local Deep Kernel Learning for Efficient Non-linear SVM Prediction](#).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Create trainer mode	List	Learner parameter option	Single Parameter	<p>Advanced learner options:</p> <ol style="list-style-type: none"> 1. Create learner using a single parameter 2. Create learner using a parameter range

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Depth of the tree	>=1	Integer	3	The depth of the locally deep SVM tree.
Lambda W	>=1.401298E-45	Float	0.1	Regularization weight for the classifier parameter Lambda W.
Lambda Theta	>=1.401298E-45	Float	0.01	Regularization weight for the classifier parameter Lambda Theta.
Lambda Theta Prime	>=1.401298E-45	Float	0.01	Regularization weight for the classifier parameter Lambda Theta prime.
Sigmoid sharpness	>=1.401298E-45	Float	1.0	The sigmoid sharpness.
Depth of the tree	[1:int.MaxValue]	ParameterRangeSettings	1; 3; 5; 7	The range for the depth of the locally deep SVM tree.
Lambda W	[1.401298E-45;3.40282347E+38]	ParameterRangeSettings	0.1; 0.01; 0.001	Range for the regularization weight for the classifier parameter Lambda W.
Lambda Theta	[1.401298E-45;3.40282347E+38]	ParameterRangeSettings	0.1; 0.01; 0.001	Range for the regularization weight for the classifier parameter Lambda Theta.
Lambda Theta Prime	[1.401298E-45;3.40282347E+38]	ParameterRangeSettings	0.1; 0.01; 0.001	Range for the regularization weight for the classifier parameter Lambda Theta prime'.
Sigmoid sharpness	[1.401298E-45;3.40282347E+38]	ParameterRangeSettings	1.0; 0.1; 0.01	The range for the sigmoid sharpness.
Feature normalizer	List	Normalizer type	Min-Max normalizer	The type of normalization to apply to learning examples.
Number of iterations	>=1	Integer	15000	Number of learning iterations.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Number of iterations	[1;int.MaxValue]	ParameterRangeSettings	10000; 15000; 20000	The range for the number of learning iterations.
<i>Random number seed</i>	Any	Integer		The seed for the random number generator that is used by the model. Leave it blank for the default.
Allow unknown categorical levels	Any	Boolean	True	If True, creates an additional level for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

Output

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained binary classification model.

See also

[Classification](#)

[A-Z Module List](#)

Two-Class Logistic Regression

3/10/2021 • 7 minutes to read • [Edit Online](#)

Creates a two-class logistic regression model

Category: [Machine Learning](#) / [Initialize Model](#) / [Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Two-Class Logistic Regression** module in Azure Machine Learning Studio (classic), to create a logistic regression model that can be used to predict two (and only two) outcomes.

Logistic regression is a well-known statistical technique that is used for modeling many kinds of problems. This algorithm is a *supervised learning* method; therefore, you must provide a dataset that already contains the outcomes to train the model.

More about logistic regression

Logistic regression is a well-known method in statistics that is used to predict the probability of an outcome, and is especially popular for classification tasks. The algorithm predicts the probability of occurrence of an event by fitting data to a logistic function. For details about this implementation, see the [Technical Notes](#) section.

In this module, the classification algorithm is optimized for dichotomous or binary variables. If you need to classify multiple outcomes, use the [Multiclass Logistic Regression](#) module.

How to configure Two-Class Logistic Regression

To train this model, you must provide a dataset that contains a label or class column. Because this module is intended for two-class problems, the label or class column must contain exactly two values.

For example, the label column might be [Voted] with possible values of "Yes" or "No". Or, it might be [Credit Risk], with possible values of "High" or "Low".

1. Add the **Two-Class Logistic Regression** module to your experiment in Studio (classic).
2. Specify how you want the model to be trained, by setting the **Create trainer mode** option.
 - **Single Parameter:** If you know how you want to configure the model, you can provide a specific set of values as arguments.
 - **Parameter Range:** If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using the [Tune Model Hyperparameters](#) module to find the optimal configuration. The trainer iterates over multiple combinations of the settings and determines the combination of values that produces the best model.
3. For **Optimization tolerance**, specify a threshold value to use when optimizing the model. If the improvement between iterations falls below the specified threshold, the algorithm is considered to have

converged on a solution, and training stops.

4. For **L1 regularization weight** and **L2 regularization weight**, type a value to use for the regularization parameters L1 and L2. A non-zero value is recommended for both.

Regularization is a method for preventing overfitting by penalizing models with extreme coefficient values. Regularization works by adding the penalty that is associated with coefficient values to the error of the hypothesis. Thus, 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.

- L1 can be applied to sparse models, which is useful when working with high-dimensional data.
- In contrast, L2 regularization is preferable for data that is not sparse.

This algorithm supports a linear combination of L1 and L2 regularization values: that is, if $x = L1$ and $y = L2$, then $ax + by = c$ defines the linear span of the regularization terms.

NOTE

Want to learn more about L1 and L2 regularization? The following article provides a discussion of how L1 and L2 regularization are different and how they affect model fitting, with code samples for logistic regression and neural network models: [L1 and L2 Regularization for Machine Learning](#)

Different linear combinations of L1 and L2 terms have been devised for logistic regression models: for example, [elastic net regularization](#). We suggest that you reference these combinations to define a linear combination that is effective in your model.

5. For **Memory size for L-BFGS**, specify the amount of memory to use for *L-BFGS* optimization.

L-BFGS stands for "limited memory Broyden-Fletcher-Goldfarb-Shanno". It is an optimization algorithm that is popular for parameter estimation. This parameter indicates 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 next step and direction. When you specify less memory, training is faster but less accurate.

6. For **Random number seed**, type an integer value. Defining a seed value is important if you want the results to be reproducible over multiple runs of the same experiment.
7. Select the **Allow unknown categorical levels** option to create an additional "unknown" level in each categorical column. If you do so, any values (levels) in the test dataset that are not available in the training dataset are mapped to this "unknown" level.
8. Add a tagged dataset to the experiment, and connect one of the [training modules](#).

- If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
- If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#) module.

NOTE

If you pass a parameter range to [Train Model](#), it will use only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and using the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified will be used throughout the sweep, even if other parameters change across a range of values.

9. Run the experiment.

Results

After training is complete:

- To see a summary of the model's parameters, together with the feature weights learned from training, right-click the output of [Train Model](#) or [Tune Model Hyperparameters](#), and select **Visualize**.
- To make predictions on new data, use the trained model and new data as input to the [Score Model](#) module.
- To perform cross-validation against a labeled data set, connect the data and the untrained model to [Cross-Validate Model](#).

Examples

For examples of how this learning algorithm is used, see the [Azure AI Gallery](#):

- [Network intrusion detection](#): Uses binary logistic regression to determine whether a case represents an intrusion.
- [Cross-Validation for Binary Classifier](#): Demonstrates the use of logistic regression in a typical experimental workflow, including model evaluation.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Usage tips

Logistic regression requires numeric variables. Therefore, when you use categorical columns as variable, Azure Machine Learning converts the values to an indicator array internally.

For dates and times, a numeric representation is used. (For more information about date time values, see [DateTime Structure \(.NET Framework\) - Remarks](#).) If you want to handle dates and times differently we suggest that you create a derived column.

Implementation details

Logistic regression assumes a *logistic distribution* of the data, where the probability that an example belongs to class 1 is the formula:

$$p(x; \beta_0, \dots, \beta_{D-1})$$

Where:

- x is a D-dimensional vector containing the values of all the features of the instance.
- p is the logistic distribution function.

- $\beta_{\{0\}}, \dots, \beta_{\{D-1\}}$ are the unknown parameters of the logistic distribution.

The algorithm tries to find the optimal values for $\beta_{\{0\}}, \dots, \beta_{\{D-1\}}$ by maximizing the log probability of the parameters given the inputs. Maximization is performed by using a popular method for parameter estimation, called [Limited Memory BFGS](#).

Research

For more information on the implementation of this algorithm, see [Scalable Training of L-1 Regularized Log-Linear Models](#), by Andrew and Gao.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Optimization tolerance	$>= \text{double.Epsilon}$	Float	0.0000001	Specify a tolerance value for the L-BFGS optimizer
L1 regularization weight	$>= 0.0$	Float	1.0	Specify the L1 regularization weight
L2 regularization weight	$>= 0.0$	Float	1.0	Specify the L2 regularization weight
Memory size for L-BFGS	$>= 1$	Integer	20	Specify the amount of memory (in MB) to use for the L-BFGS optimizer
Random number seed	Any	Integer		Type a value to seed the random number generator used by the model. Leave it blank for the default.
Allow unknown categorical levels	Any	Boolean	True	Indicate whether an additional level should be created for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

Output

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained classification model

See also

[Classification](#)

[Multiclass Logistic Regression](#)

[A-Z Module List](#)

Two-Class Neural Network

3/10/2021 • 9 minutes to read • [Edit Online](#)

Creates a binary classifier using a neural network algorithm

Category: [Machine Learning / Initialize Model / Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Two-Class Neural Network** module in Azure Machine Learning Studio (classic), to create a neural network model that can be used to predict a target that has only two values.

Classification using neural networks is a supervised learning method, and therefore requires a *tagged dataset*, which includes a label column. For example, you could use this neural network model to predict binary outcomes such as whether or not a patient has a certain disease, or whether a machine is likely to fail within a specified window of time.

After you define the model, train it by providing a tagged dataset and the model as an input to [Train Model](#) or to [Tune Model Hyperparameters](#). The trained model can then be used to predict values for new inputs.

More about neural networks

A neural network is a set of interconnected layers. The inputs are the first layer, and are connected to an output layer by an acyclic graph comprised of weighted edges and nodes.

Between the input and output layers you can insert multiple hidden layers. Most predictive tasks can be accomplished easily with only one or a few hidden layers. However, recent research has shown that deep neural networks (DNN) with many layers can be very effective in complex tasks such as image or speech recognition. The successive layers are used to model increasing levels of semantic depth.

The relationship between inputs and outputs is learned from training the neural network on the input data. The direction of the graph proceeds from the inputs through the hidden layer and to the output layer. All nodes in a layer are connected by the weighted edges to nodes in the next layer.

To compute the output of the network for a particular input, a value is calculated at each node in the hidden layers and in the output layer. The value is set by calculating the weighted sum of the values of the nodes from the previous layer. An activation function is then applied to that weighted sum.

How to configure Two-Class Neural Network

1. Add the **Two-Class Neural Network** module to your experiment in Studio (classic). You can find this module under **Machine Learning, Initialize**, in the **Classification** category.
2. Specify how you want the model to be trained, by setting the **Create trainer mode** option.
 - **Single Parameter:** Choose this option if you already know how you want to configure the model.

- **Parameter Range:** Choose this option if you are not sure of the best parameters. Then, specify a range of values and use the [Tune Model Hyperparameters](#) module to iterate over the combinations and find the optimal configuration.

3. For **Hidden layer specification**, select the type of network architecture to create.

- **Fully-connected case:** Uses the default neural network architecture, defined for two-class neural networks as follows:
 - Has one hidden layer.
 - The output layer is fully connected to the hidden layer, and the hidden layer is fully connected to the input layer.
 - The number of nodes in the input layer equals the number of features in the training data.
 - The number of nodes in the hidden layer is set by the user. The default value is 100.
 - The number of nodes equals the number of classes. For a two-class neural network, this means that all inputs must map to one of two nodes in the output layer.
- **Custom definition script:** Choose this option to create a custom neural network architecture, using the [Net# language](#). With this option, you can define the number of hidden layers, their connections, and the mappings between layers.

After selecting the custom script option, in the **Neural network definition** text box, type or paste Net# statements that define the network. For examples, see [Guide to the Net# Neural Networks Specification Language](#).

4. If you are not using the script option, use **Number of hidden nodes**, and type the number of hidden nodes. The default is one hidden layer with 100 nodes.
5. For **Learning rate**, define the size of the step taken at each iteration, before correction. A larger value for learning rate can cause the model to converge faster, but it can overshoot local minima.
6. For **Number of learning iterations**, specify the maximum number of times the algorithm should process the training cases.
7. For **The initial learning weights diameter**, specify the node weights at the start of the learning process.
8. For **The momentum**, specify a weight to apply during learning to nodes from previous iterations
9. In **The type of normalizer**, select a method to use for feature normalization. The following normalization methods are supported:
 - **Binning normalizer:** The binning normalizer creates bins of equal size, and then normalizes every value in each bin, dividing by the total number of bins.
 - **Gaussian normalizer:** The Gaussian normalizer rescales the values of each feature to have mean 0 and variance 1. This is done by computing the mean and the variance of each feature. For each instance, the mean value is subtracted, and the result divided by the square root of the variance (the standard deviation).
 - **Min-max normalizer:** The min-max normalizer linearly rescales every feature to the [0,1] interval.

Rescaling to the [0,1] interval is done by shifting the values of each feature so that the minimal value is 0, and then dividing by the new maximal value (which is the difference between the original maximal and minimal values).

- **Do not normalize:** No normalization is performed.
10. Select the **Shuffle examples** option to shuffle cases between iterations. If you deselect this option, cases are processed in exactly the same order each time you run the experiment.
11. For **Random number seed**, type a value to use as the seed.
- Specifying a seed value is useful when you want to ensure repeatability across runs of the same experiment. Otherwise, a system clock value is used as the seed, which can cause slightly different results each time you run the experiment.
12. Select the **Allow unknown categorical levels** option to create a grouping for unknown values in the training and validation sets. The model might be less precise on known values but provide better predictions for new (unknown) values.
- If you deselect this option, the model can accept only the values contained in the training data.
13. Add a tagged dataset to the experiment, and connect one of the [training modules](#).
- If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
 - If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#) module.

NOTE

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value is used throughout the sweep, even if other parameters change across a range of values.

14. Run the experiment.

Results

After training is complete:

- To see a summary of the model's parameters, together with the feature weights learned from training, and other parameters of the neural network, right-click the output of [Train Model](#) or [Tune Model Hyperparameters](#), and select [Visualize](#).
- To save a snapshot of the trained model, right-click the **Trained model** output and select [Save As Trained Model](#). This model is not updated on successive runs of the same experiment.
- To perform cross-validation against a labeled data set, connect the untrained model to [Cross-Validate Model](#).

Examples

For examples of how this learning algorithm is used, see the [Azure AI Gallery](#). These experiments are related and described in a single document that progresses from basic to advanced configurations:

- [Deep Neural networks sample \(part A\)](#)
- [Deep Neural networks sample \(part B\)](#)
- [Deep Neural networks sample \(part C\)](#)

- Deep Neural networks sample (part D)

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

More about Net#

In Azure Machine Learning Studio (classic), you can customize the architecture of a neural network model by using the Net# language. Customizations supported by the Net# language include:

- Specifying the number of hidden layers and the number of nodes in each layer
- Specifying mappings between layers
- Defining convolutions and weight-sharing bundles
- Choosing the activation function

A neural network model is defined by the structure of its graph, which includes these attributes:

- The number of hidden layers
- The number of nodes in each hidden layer
- How the layers are connected
- Which activation function is used
- Weights on the graph edges

IMPORTANT

The overall structure of the graph, as well as the activation function, can be specified by the user. However, the weights on the edges cannot be specified, and must be learned when training the neural network on the input data.

In general, the network has these defaults:

- The first layer is always the input layer.
- The last layer is always the output layer.
- The number of nodes in the output layer should be equal to the number of classes.

You can define any number of intermediate layers (sometimes called hidden layers, because they are contained within the model, and they are not directly exposed as endpoints).

The Net# reference guide explains the syntax and provides sample network definitions. It explains how you can use Net# to add hidden layers and define the way that the different layers interact with each other.

For example, the following script uses the `auto` keyword, which sets the number of features automatically for input and output layers, and uses the default values for the hidden layer.

```
input Data auto;
hidden Hidden auto from Data all;
output Result auto from Hidden all;
```

For additional script examples, see [Guide to the Net# Neural Networks Specification Language](#).

TIP

Neural networks can be computationally expensive, due to a number of hyperparameters and the introduction of custom network topologies. Although in many cases neural networks produce better results than other algorithms, obtaining such results may involve a fair amount of sweeping (iterations) over hyperparameters.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Hidden layer specification	List	Neural Network Topology	Fully-connected case	Specify the architecture of the hidden layer or layers
The initial learning weights diameter	$\geq \text{double.Epsilon}$	Float	0.1	Specify the node weights at the start of the learning process
Learning rate	$[\text{double.Epsilon}; 1.0]$	Float	0.1	Specify the size of each step in the learning process
The momentum	$[0.0; 1.0]$	Float	0.0	Specify a weight to apply during learning to nodes from previous iterations
Neural network definition	Any	StreamReader		When you select Custom definition script , type a valid script expression on each line to define the layers, nodes, and behavior of a custom neural network
The type of normalizer	List	Normalization Method	Min-Max normalizer	Select the type of normalization to apply to learning examples
Number of learning iterations	≥ 1	Integer	100	Specify the number of iterations performed during learning
Shuffle examples	Any	Boolean	true	Select this option to change the order of instances between learning iterations

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Random number seed	Any	Integer		Specify a numeric seed to use for random number generation. Leave it blank to use the default seed.
Allow unknown categorical levels	Any	Boolean	True	Indicated whether an additional level should be created for unknown categories. If the test dataset contains categories that are not present in the training dataset, they are mapped to this unknown level.

Output

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained binary classification model

See also

[Classification](#)

[Neural Network Regression](#)

[Multiclass Neural Network](#)

[A-Z Module List](#)

Two-Class Support Vector Machine

3/10/2021 • 5 minutes to read • [Edit Online](#)

Creates a binary classification model using the Support Vector Machine algorithm

Category: [Machine Learning / Initialize Model / Classification](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Two-Class Support Vector Machine** module in Azure Machine Learning Studio (classic), to create a model that is based on the support vector machine algorithm.

Support vector machines (SVMs) are a well-researched class of supervised learning methods. This particular implementation is suited to prediction of two possible outcomes, based on either continuous or categorical variables.

After defining the model parameters, train the model by using one of the [training modules](#), and providing a *tagged dataset* that includes a label or outcome column.

More about support vector machines

Support vector machines are among the earliest of machine learning algorithms, and SVM models have been used in many applications, from information retrieval to text and image classification. SVMs can be used for both classification and regression tasks.

This SVM model is a supervised learning model that requires labeled data. In the training process, the algorithm analyzes input data and recognizes patterns in a multi-dimensional feature space called the *hyperplane*. All input examples are represented as points in this space, and are mapped to output categories in such a way that categories are divided by as wide and clear a gap as possible.

For prediction, the SVM algorithm assigns new examples into one category or the other, mapping them into that same space.

How to configure Two-Class Support Vector Machine

For this model type, it is recommended that you normalize the dataset before using it to train the classifier.

1. Add the **Two-Class Support Vector Machine** module to your experiment in Studio (classic).

2. Specify how you want the model to be trained, by setting the **Create trainer mode** option.

- **Single Parameter:** If you know how you want to configure the model, you can provide a specific set of values as arguments.
- **Parameter Range:** If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using the [Tune Model Hyperparameters](#) module to find the optimal configuration. The trainer iterates over multiple combinations of the settings and

determines the combination of values that produces the best model.

3. For **Number of iterations**, type a number that denotes the number of iterations used when building the model.

This parameter can be used to control trade-off between training speed and accuracy.

4. For **Lambda**, type a value to use as the weight for L1 regularization.

This regularization coefficient can be used to tune the model. Larger values penalize more complex models.

5. Select the option, **Normalize features**, if you want to normalize features before training.

If you apply normalization, before training, data points are centered at the mean and scaled to have one unit of standard deviation.

6. Select the option, **Project to the unit sphere**, to normalize coefficients.

Projecting values to unit space means that before training, data points are centered at 0 and scaled to have one unit of standard deviation.

7. In **Random number seed**, type an integer value to use as a seed if you want to ensure reproducibility across runs. Otherwise, a system clock value is used as a seed, which can result in slightly different results across runs.

8. Select the option, **Allow unknown category**, to create a group for unknown values in the training or validation sets. In this case, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.

If you deselect it, the model can accept only the values that are contained in the training data.

9. Connect a labeled dataset, and one of the [training modules](#):

- If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
- If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#).

NOTE

If you pass a parameter range to [Train Model](#), it will use only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and using the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified will be used throughout the sweep, even if other parameters change across a range of values.

10. Run the experiment.

Results

After training is complete:

- To see a summary of the model's parameters, together with the feature weights learned from training,, right-click the output of [Train Model](#) or [Tune Model Hyperparameters](#), and select **Visualize**.
- To use the trained models to make predictions, connect the trained model to the [Score Model](#) module.
- To perform cross-validation against a labeled data set, connect the untrained model and the dataset to [Cross-Validate Model](#).

Examples

For examples of how this learning algorithm is used, see the [Azure AI Gallery](#):

- [Direct marketing](#): Uses an SVM model to classify customers by appetency.
- [Credit risk prediction](#): Uses SVM for assessing credit risk.
- [Compare Multiclass Classifiers](#): Uses an SVM model for handwriting recognition.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Usage tips

For this model type, it is recommended that you normalize the dataset before using it to train the classifier.

Although recent research has developed algorithms that have higher accuracy, this algorithm can work well on simple data sets when your goal is speed over accuracy. If you do not get the desired results by using **Two-Class Support Vector Model**, try one of these classification methods:

- [Multiclass Logistic Regression](#)
- [Two-Class Boosted Decision Tree](#)

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Number of iterations	$>=1$	Integer	1	The number of iterations
Lambda	$>=\text{double.Epsilon}$	Float	0.001	Weight for L1 regularization. Using a non-zero value avoids overfitting the model to the training dataset.
Normalize features	Any	Boolean	True	If True, normalize the features.
Project to the unit-sphere	Any	Boolean	False	If True, project the features to a unit circle.
Random number seed	Any	Integer		The seed for the random number generator used by the model. Leave it blank for the default.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Allow unknown categorical levels	Any	Boolean	True	If True, creates an additional level for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level.

Output

NAME	TYPE	DESCRIPTION
Untrained model	Data Table	An untrained binary classification model.

See also

[Classification](#)

[A-Z Module List](#)

Clustering modules

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that support creation of clustering models.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

What is clustering?

Clustering, in machine learning, is a method of grouping data points into similar clusters. It is also called *segmentation*.

Over the years, many clustering algorithms have been developed. Almost all clustering algorithms use the features of individual items to find similar items. For example, you might apply clustering to find similar people by demographics. You might use clustering with text analysis to group sentences with similar topics or sentiment.

Clustering is called a non-supervised learning technique because it can be used in unlabeled data. Indeed, clustering is a useful first step for discovering new patterns, and requires little prior knowledge about how the data might be structured or how items are related. Clustering is often used for exploration of data prior to analysis with other more predictive algorithms.

How to create a clustering model

In Machine Learning Studio (classic), you can use clustering with either labeled or unlabeled data.

- In unlabeled data, the clustering algorithm determines which data points are closest together, and creates clusters around a central point, or centroid. You can then use the cluster ID as a temporary label for the group of data.
- If the data has labels, you can use the label to drive the number of clusters, or use the label as just another feature.

After you have configured the clustering algorithm, you train it on data by using either the [Train Clustering Model](#) or [Sweep Clustering](#) modules.

When the model is trained, use it to predict cluster membership for new data points. For example, if you have used clustering to group customers by purchasing behavior, you can use the model to predict the purchasing behavior of new customers.

List of modules

The clustering category includes this module:

- [K-Means Clustering](#): Configures and initializes a K-means clustering model.

Related tasks

To use a different clustering algorithm, or create a custom clustering model by using R, see these topics:

- [Execute R Script](#)
- [Create R Model](#)

Examples

For examples of clustering in action, see the [Azure AI Gallery](#).

See these articles for help choosing an algorithm:

- [Machine learning algorithm cheat sheet for Azure Machine Learning Studio \(classic\)](#)
Provides a graphical decision chart to guide you through the selection process.
- [How to choose Azure Machine Learning algorithms for clustering, classification, or regression](#)
Explains in greater detail the different types of machine learning algorithms, and how they're used.

See also

- [Regression](#)
- [Classification](#)
- [Text Analytics](#)
- [Image classification using OpenCV](#)

K-Means Clustering

3/10/2021 • 13 minutes to read • [Edit Online](#)

Configures and initializes a K-means clustering model

Category: [Machine Learning / Initialize Model / Clustering](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **K-Means Clustering** module in Azure Machine Learning Studio (classic) to create an untrained K-means clustering model.

K-means is one of the simplest and the best known *unsupervised* learning algorithms, and can be used for a variety of machine learning tasks, such as [detecting abnormal data](#), clustering of text documents, and analysis of a dataset prior to using other classification or regression methods. To create a clustering model, you add this module to your experiment, connect a dataset, and set parameters such as the number of clusters you expect, the distance metric to use in creating the clusters, and so forth.

After you have configured the module hyperparameters, connect the untrained model to the [Train Clustering Model](#) or the [Sweep Clustering](#) modules to train the model on the input data that you provide. Because the K-means algorithm is an unsupervised learning method, a label column is optional.

- If your data includes a label, you can use the label values to guide selection of the clusters and optimize the model.
- If your data has no label, the algorithm creates clusters representing possible categories, based solely on the data.

TIP

If your training data has labels, consider using one of the supervised [classification](#) methods provided in Azure Machine Learning. For example, you might compare the results of clustering to the results when using one of the multiclass decision tree algorithms.

Understanding k-means clustering

In general, clustering uses iterative techniques to group cases in a dataset into clusters that contain similar characteristics. These groupings are useful for exploring data, identifying anomalies in the data, and eventually for making predictions. Clustering models can also help you identify relationships in a dataset that you might not logically derive by browsing or simple observation. For these reasons, clustering is often used in the early phases of machine learning tasks, to explore the data and discover unexpected correlations.

When you configure a clustering model using the k-means method, you must specify a target number k indicating the number of *centroids* you want in the model. The centroid is a point that is representative of each cluster. The K-means algorithm assigns each incoming data point to one of the clusters by minimizing the

within-cluster sum of squares.

When processing the training data, the K-means algorithm begins with an initial set of randomly chosen centroids, which serve as starting points for each cluster, and applies Lloyd's algorithm to iteratively refine the locations of the centroids. The K-means algorithm stops building and refining clusters when it meets one or more of these conditions:

- The centroids stabilize, meaning that cluster assignments for individual points no longer change and the algorithm has converged on a solution.
- The algorithm completed running the specified number of iterations.

After completing the training phase, you use the [Assign Data to Clusters](#) module to assign new cases to one of the clusters that was found by the k-means algorithm. Cluster assignment is performed by computing the distance between the new case and the centroid of each cluster. Each new case is assigned to the cluster with the nearest centroid.

How to configure K-Means Clustering

1. Add the **K-Means Clustering** module to your experiment.
2. Specify how you want the model to be trained, by setting the **Create trainer mode** option.
 - **Single Parameter**: If you know the exact parameters you want to use in the clustering model, you can provide a specific set of values as arguments.
 - **Parameter Range**: If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using the [Sweep Clustering](#) module to find the optimal configuration.

The trainer iterates over multiple combinations of the settings you provided and determine the combination of values that produces the optimal clustering results.

3. For **Number of Centroids**, type the number of clusters you want the algorithm to begin with.

The model is not guaranteed to produce exactly this number of clusters. The algorithm starts with this number of data points and iterates to find the optimal configuration, as described in the [Technical Notes](#) section.

If you are performing a parameter sweep, the name of the property changes to **Range for Number of Centroids**. You can use the **Range Builder** to specify a range, or you can type a series of numbers representing different numbers of clusters to create when initializing each model.

4. The properties **Initialization** or **Initialization for sweep** are used to specify the algorithm that is used to define the initial cluster configuration.
 - **First N**: Some initial number of data points are chosen from the data set and used as the initial means.
Also called the *Forgy method*.
 - **Random**: The algorithm randomly places a data point in a cluster and then computes the initial mean to be the centroid of the cluster's randomly assigned points.
Also called the *random partition* method.
 - **K-Means++**: This is the default method for initializing clusters.

The **K-means ++** algorithm was proposed in 2007 by David Arthur and Sergei Vassilvitskii to avoid poor clustering by the standard k-means algorithm. **K-means ++** improves upon standard

K-means by using a different method for choosing the initial cluster centers.

- **K-Means++ Fast:** A variant of the K-means ++ algorithm that was optimized for faster clustering.
- **Evenly:** Centroids are located equidistant from each other in the d-Dimensional space of n data points.
- **Use label column:** The values in the label column are used to guide the selection of centroids.

5. For **Random number seed**, optionally type a value to use as the seed for the cluster initialization. This value can have a significant effect on cluster selection.

If you use a parameter sweep, you can specify that multiple initial seeds be created, to look for the best initial seed value. For **Number of seeds to sweep**, type the total number of random seed values to use as starting points.

6. For **Metric**, choose the function to use for measuring the distance between cluster vectors, or between new data points and the randomly chosen centroid. Azure Machine Learning supports the following cluster distance metrics:

- **Euclidean:** The Euclidean distance is commonly used as a measure of cluster scatter for K-means clustering. This metric is preferred because it minimizes the mean distance between points and the centroids.
- **Cosine:** The cosine function is used to measure cluster similarity. Cosine similarity is useful in cases where you do not care about the length of a vector, only its angle.

7. For **Iterations**, type the number of times the algorithm should iterate over the training data before finalizing the selection of centroids.

You can adjust this parameter to balance accuracy vs. training time.

8. For **Assign label mode**, choose an option that specifies how a label column, if present in the dataset, should be handled.

Because K-means clustering is an unsupervised machine learning method, labels are optional. However, if your dataset already has a label column, you can use those values to guide selection of the clusters, or you can specify that the values be ignored.

- **Ignore label column:** The values in the label column are ignored and are not used in building the model.
- **Fill missing values:** The label column values are used as features to help build the clusters. If any rows are missing a label, the value is imputed by using other features.
- **Overwrite from closest to center:** The label column values are replaced with predicted label values, using the label of the point that is closest to the current centroid.

9. Train the model.

- If you set **Create trainer mode to Single Parameter**, add a tagged dataset and train the model by using the [Train Clustering Model](#) module.
- If you set **Create trainer mode to Parameter Range**, add a tagged dataset and train the model using [Sweep Clustering](#). You can use the model trained using those parameters, or you can make a note of the parameter settings to use when configuring a learner.

Results

After you have finished configuring and training the model, you have a model that you can use to generate

scores. However, there are multiple ways to train the model, and multiple ways to view and use the results:

Capture a snapshot of the model in your workspace

- If you used the [Train Clustering Model](#) module
 1. Right-click the [Train Clustering Model](#) module.
 2. Select **Trained model** and then click **Save as Trained Model**.
- If you used the [Sweep Clustering](#) module to train the model
 1. Right-click the [Sweep Clustering](#) module.
 2. Select **Best Trained model** and then click **Save as Trained Model**.

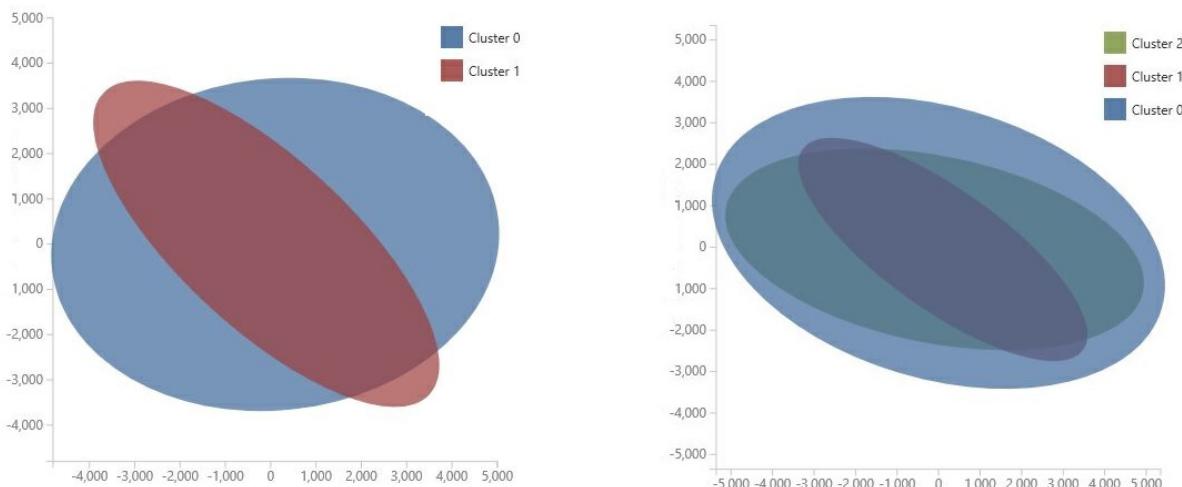
The saved model will represent the training data at the time you saved the model. If you later update the training data used in the experiment, it will not update the saved model.

See a visual representation of the clusters in the model

- If you used the [Train Clustering Model](#) module
 1. Right-click the module, and select **Results dataset**.
 2. Select **Visualize**.
- If you used the [Sweep Clustering](#) module
 1. Add an instance of the [Assign Data to Clusters](#) module and generate scores using the **Best Trained model**.
 2. Right-click the [Assign Data to Clusters](#) module, select **Results dataset**, and select **Visualize**.

The chart is generated by using [Principal Component Analysis](#), which is a technique in data science for compressing the feature space of a model. The chart shows some set of features, compressed into two dimensions, that best characterize the difference between the clusters. By visually reviewing the general size of the feature space for each cluster and how much the clusters overlap, you can get an idea of how well your model might perform.

For example, the following PCA charts represent the results from two models trained using the same data: the first was configured to output two clusters, and the second was configured to output three clusters. From these charts, you can see that increasing the number of clusters did not necessarily improve separation of the classes.



TIP

Use the [Sweep Clustering](#) module to choose the optimal set of hyperparameters, including the random seed and number of starting centroids.

See the list of data points and the clusters they belong to

There are two options for viewing the dataset with results, depending on how you trained the model:

- If you used the [Sweep Clustering](#) module to train the model
 1. Use the checkbox in the **Sweep Clustering** module to specify whether you want to see the input data together with the results, or see just the results.
 2. When training is complete, right-click the module, and select **Results dataset** (output number 2)
 3. Click **Visualize**.
- If you used the [Train Clustering Model](#) module
 1. Add the [Assign Data to Clusters](#) module and connect the trained model to the left-hand input. Connect a dataset to the right-hand input.
 2. Add the [Convert to Dataset](#) module to your experiment and connect it to the output of **Assign Data to Clusters**.
 3. Use the checkbox in the **Assign Data to Clusters** module to specify whether you want to see the input data together with the results, or see just the results.
 4. Run the experiment, or run just the **Convert to Dataset** module.
 5. Right-click **Convert to Dataset**, select **Results dataset**, and click **Visualize**.

The output contains the input data columns first, if you included them, and the following columns for each row of input data:

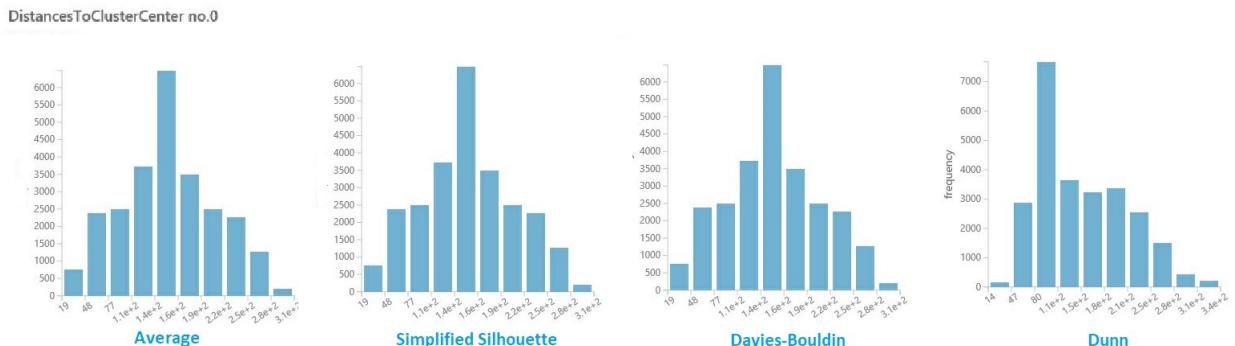
- **Assignment:** The assignment is a value between 1 and n , where n is the total number of clusters in the model. Each row of data can be assigned to only one cluster.
- **DistancesToClusterCenter no.n:** This value measures the distance from the current data point to the centroid for the cluster. A separate column in output for each cluster in the trained model.

The values for cluster distance are based on the distance metric you selected in the option, **Metric for measuring cluster result**. Even if you perform a parameter sweep on the clustering model, only one metric can be applied during the sweep. If you change the metric, you might get different distance values.

Visualize intra-cluster distances

In the dataset of results from the previous section, click the column of distances for each cluster. Studio (classic) displays a histogram that visualizes the distribution of distances for points within the cluster.

For example, the following histograms show the distribution of cluster distances from the same experiment, using four different metrics. All other settings for the parameter sweep were the same. Changing the metric resulted in a different number of clusters in one model.



In general, you should choose a metric that maximizes the distance between data points in different classes, and minimizes distances within a class. You can use the precomputed means and other values in the **Statistics** pane to guide you in this decision.

TIP

You can extract means and other values used in visualizations by using the [PowerShell module for Azure Machine Learning](#).

Or use the [Execute R Script](#) module to compute a custom distance matrix.

Tips for generating the best clustering model

It is known that the **seeding** process used during clustering can significantly affect the model. Seeding means the initial placement of points into potential centroids.

For example, if the dataset contains many outliers, and an outlier is chosen to seed the clusters, no other data points would fit well with that cluster and the cluster could be a singleton: that is, a cluster with only one point.

There are various ways to avoid this problem:

- Use a parameter sweep to change the number of centroids and try multiple seed values.
- Create multiple models, varying the metric or iterating more.
- Use a method such as PCA to find variables that have a detrimental effect on clustering. See the [Find similar companies](#) sample for a demonstration of this technique.

In general, with clustering models, it is possible that any given configuration will result in a locally optimized set of clusters. In other words, the set of clusters returned by the model suits only the current data points, and is not generalizable to other data. If you used a different initial configuration, the K-means method might find a different, perhaps superior, configuration.

IMPORTANT

We recommend that you always experiment with the parameters, create multiple models, and compare the resulting models.

Examples

For examples of how K-means clustering is used in Azure Machine Learning, see these experiments in the [Azure AI Gallery](#):

- [Group iris data](#): Compares the results of **K-Means Clustering** and **Multiclass Logistic Regression** for a classification task.
- [Color Quantization sample](#): Builds multiple K-means models with different parameters to find the optimum image compression.
- [Clustering: Similar Companies](#): Varies the numbers of centroids to find groups of similar companies in the S&P500.

Technical notes

Given a specific number of clusters (K) to find for a set of D -dimensional data points with N data points, the K-means algorithm builds the clusters as follows:

1. The module initializes a K -by- D array with the final centroids that define the K clusters found.
2. By default, the module assigns the first K data points in order to the K clusters.
3. Starting with an initial set of K centroids, the method uses Lloyd's algorithm to iteratively refine the

locations of the centroids.

4. The algorithm terminates when the centroids stabilize or when a specified number of iterations are completed.
5. A similarity metric (by default, Euclidean distance) is used to assign each data point to the cluster that has the closest centroid.

WARNING

- If you pass a parameter range to [Train Clustering Model](#), it uses only the first value in the parameter range list.
- If you pass a single set of parameter values to the [Sweep Clustering](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.
- If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Number of Centroids	≥ 2	Integer	2	Number of Centroids
Metric	List (subset)	Metric	Euclidean	Selected metric
Initialization	List	Centroid initialization method	K-Means++	Initialization algorithm
Iterations	≥ 1	Integer	100	Number of iterations

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ICluster interface	Untrained K-Means clustering model

Exceptions

For a list of all exceptions, see [Machine Learning Module Error Codes](#).

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.

See also

[Clustering](#)
[Assign Data to Clusters](#)
[Train Clustering Model](#)
[Sweep Clustering](#)

Regression modules

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that support creation of regression models.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

More about regression

Regression is a methodology used widely in fields ranging from engineering to education. For example, you might use regression to predict the value of a house based on regional data, or to create projections about future enrollment.

Regression tasks are supported in many tools: for example, Excel provides "What If" analysis, forecasting over time, and the Analysis ToolPak for traditional regression.

The modules for regression in Machine Learning Studio (classic) each incorporate a different method, or algorithm, for regression. In general, a regression algorithm tries to learn the value of a function for a particular instance of data. You might predict someone's height by using a height function, or predict the probability of hospital admission based on medical test values.

Regression algorithms can incorporate input from multiple features, by determining the contribution of each feature of the data to the regression function.

How to create a regression model

First, select the regression algorithm that meets your needs and suits your data. For help, see these topics:

- [Machine learning algorithm cheat sheet for Azure Machine Learning](#)
Provides a graphical decision chart to guide you through the selection process.
- [How to choose Azure Machine Learning algorithms for clustering, classification, or regression](#)

Explains in greater detail the different types of machine learning algorithms, and how they're used.

Add training data. Be sure to consult the module reference for each algorithm in advance, to determine if the training data has any special requirements, other than a numeric outcome.

To train the model, run the experiment. After the regression algorithm has learned from the labeled data, you can use the function it learned to make predictions on new data.

List of modules

- [Bayesian Linear Regression](#): Creates a Bayesian linear regression model.
- [Boosted Decision Tree Regression](#): Creates a regression model by using the Boosted Decision Tree algorithm.
- [Decision Forest Regression](#): Creates a regression model by using the decision forest algorithm.
- [Fast Forest Quantile Regression](#): Creates a quantile regression model.

- [Linear Regression](#): Creates a linear regression model.
- [Neural Network Regression](#): Creates a regression model by using a neural network algorithm.
- [Ordinal Regression](#): Creates an ordinal regression model.
- [Poisson Regression](#): Creates a regression model that assumes data has a Poisson distribution.

Examples

For examples of regression in action, see the [Azure AI Gallery](#).

See also

- [Regression](#)
- [Classification](#)
- [Clustering](#)
- [Text Analytics](#)
- [OpenCV Library Modules](#)

Bayesian Linear Regression

3/10/2021 • 2 minutes to read • [Edit Online](#)

Creates a Bayesian linear regression model

Category: [Machine Learning](#) / [Initialize Model](#) / [Regression](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Bayesian Linear Regression** module in Azure Machine Learning Studio (classic), to define a regression model based on Bayesian statistics.

After you have defined the model parameters, you must train the model using a tagged dataset and the [Train Model](#) module. The trained model can then be used to make predictions. Alternatively, the untrained model can be passed to [Cross-Validate Model](#) for cross-validation against a labeled data set.

More about Bayesian regression

In statistics, the *Bayesian* approach to regression is often contrasted with the *frequentist* approach.

The Bayesian approach uses linear regression supplemented by additional information in the form of a prior probability distribution. Prior information about the parameters is combined with a likelihood function to generate estimates for the parameters.

In contrast, the frequentist approach, represented by standard least-square linear regression, assumes that the data contains sufficient measurements to create a meaningful model.

For more information about the research behind this algorithm, see the links in the [Technical Notes](#) section.

How to configure Bayesian Regression

1. Add the **Bayesian Linear Regression** module to your experiment. You can find the this module under **Machine Learning, Initialize**, in the **Regression** category.
2. **Regularization weight:** Type a value to use for regularization. Regularization is used to prevent overfitting. This weight corresponds to L2. For more information, see the [Technical Notes](#) section.
3. **Allow unknown categorical levels:** Select this option to create a grouping for unknown values. The model can accept only the values contained in the training data. The model might be less precise on known values but provide better predictions for new (unknown) values.
4. Connect a training dataset, and one of the training modules. This model type has no parameters that can be changed in a parameter sweep, so although you can train the model using [Tune Model Hyperparameters](#), it cannot automatically optimize the model.
5. Select the single numeric column that you want to model or predict.

6. Run the experiment.

Results

After training is complete:

- To see a summary of the model's parameters, right-click the output of the [Train Model](#) module and select **Visualize**.
- To create predictions, use the trained model as an input to [Score Model](#).

Examples

For examples of regression models, see the [Azure AI Gallery](#).

- [Compare Regression Models sample](#): Contrasts several different kinds of regression models.

Technical notes

- The use of the lambda coefficient is described in detail in this textbook on machine learning: [Pattern Recognition and Machine Learning](#), Christopher Bishop, Springer-Verlag, 2007.
- This article is available as a PDF download from the [Microsoft Research site](#): [Bayesian Regression and Classification](#)

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Regularization weight	<code>>=double.Epsilon</code>	Float	1.0	Type a constant to use in regularization. The constant represents the ratio of the precision of weight prior to the precision of noise.
Allow unknown categorical levels	Any	Boolean	true	If true creates an additional level for each categorical column. Any levels in the test dataset not available in the training dataset are mapped to this additional level.

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained Bayesian linear regression model

See also

[A-Z Module List](#)

[Regression](#)

Boosted Decision Tree Regression

3/24/2021 • 7 minutes to read • [Edit Online](#)

Creates a regression model using the Boosted Decision Tree algorithm

Category: [Machine Learning / Initialize Model / Regression](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Boosted Decision Tree Regression** module in Azure Machine Learning Studio (classic), to create an ensemble of regression trees using boosting. *Boosting* means that each tree is dependent on prior trees. The algorithm learns by fitting the residual of the trees that preceded it. Thus, boosting in a decision tree ensemble tends to improve accuracy with some small risk of less coverage.

This regression method is a supervised learning method, and therefore requires a *labeled dataset*. The label column must contain numerical values.

NOTE

Use this module only with datasets that use numerical variables.

After you have defined the model, train it by using the [Train Model](#) or [Tune Model Hyperparameters](#) modules.

TIP

Want to know more about the trees that were created? After the model has been trained, right-click the output of the [Train Model](#) module (or [Tune Model Hyperparameters](#) module) and select **Visualize** to see the tree that was created on each iteration. You can drill down into the splits for each tree and see the rules for each node.

More about boosted regression trees

Boosting is one of several classic methods for creating ensemble models, along with bagging, random forests, and so forth. In Azure Machine Learning Studio (classic), boosted decision trees use 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 in each step and correct for it in the next. Thus the prediction model is actually an ensemble of weaker prediction models.

In regression problems, boosting builds a series of trees in a step-wise fashion, and then selects the optimal tree using an arbitrary differentiable loss function.

For additional information, see these articles:

- https://wikipedia.org/wiki/Gradient_boosting#Gradient_tree_boosting

This Wikipedia article on gradient boosting provides some background on boosted trees.

- <https://research.microsoft.com/apps/pubs/default.aspx?id=132652>

Microsoft Research: From RankNet to LambdaRank to LambdaMART: An Overview. By J.C. Burges.

The gradient boosting method can also be used for classification problems by reducing them to regression with a suitable loss function. For more information about the boosted trees implementation for classification tasks, see [Two-Class Boosted Decision Tree](#).

How to configure Boosted Decision Tree Regression

1. Add the **Boosted Decision Tree** module to your experiment. You can find this module under **Machine Learning, Initialize**, under the **Regression** category.
2. Specify how you want the model to be trained, by setting the **Create trainer mode** option.
 - **Single Parameter:** Select this option if you know how you want to configure the model, and provide a specific set of values as arguments.
 - **Parameter Range:** Select this option if you are not sure of the best parameters, and want to run a parameter sweep. Select a range of values to iterate over, and the [Tune Model Hyperparameters](#) iterates over all possible combinations of the settings you provided to determine the hyperparameters that produce the optimal results.
3. **Maximum number of leaves per tree:** Indicate the maximum number of terminal nodes (leaves) that can be created in any tree.

By increasing this value, you potentially increase the size of the tree and get better precision, at the risk of overfitting and longer training time.

4. **Minimum number of samples per leaf node:** Indicate the minimum number of cases required to create any terminal node (leaf) in a tree.

By increasing this value, you increase the threshold for creating new rules. For example, with the default value of 1, even a single case can cause a new rule to be created. If you increase the value to 5, the training data would have to contain at least 5 cases that meet the same conditions.

5. **Learning rate:** Type a number between 0 and 1 that defines the step size while learning. The learning rate 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 on the best solution.

6. **Number of trees constructed:** Indicate the total number of decision trees to create in the ensemble. By creating more decision trees, you can potentially get better coverage, but training time increases.

This value also controls the number of trees displayed when visualizing the trained model. If you want to see or print a single tree, you can set the value to 1; however, this means that only one tree is produced (the tree with the initial set of parameters) and no further iterations are performed.

7. **Random number seed:** Type an optional non-negative integer to use as the random seed value. Specifying a seed ensures reproducibility across runs that have the same data and parameters.

By default, the random seed is set to 0, which means the initial seed value is obtained from the system clock.

8. **Allow unknown categorical levels:** Select this option to create a group for unknown values in the training and validation sets. If you deselect this option, the model can accept only the values that are contained in the training data. The model might be less precise for known values, but it can provide better

predictions for new (unknown) values.

9. Add a training dataset, and one of the training modules:

- If you set **Create trainer mode** option to **Single Parameter**, use the [Train Model](#) module.
- If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#) module.

NOTE

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value is used throughout the sweep, even if other parameters change across a range of values.

10. Run the experiment.

Results

After training is complete:

- To see the tree that was created on each iteration, right-click [Train Model](#) module and select **Trained model** to visualize. If you use [Tune Model Hyperparameters](#), right click the module and select **Trained best model** to visualize the best model.

Click each tree to drill down into the splits and see the rules for each node.

- To use the model for scoring, connect it to [Score Model](#), to predict values for new input examples.
- To save a snapshot of the trained model, right-click the **Trained model** output of the training module and select **Save As**. The copy of the trained model that you save is not updated on successive runs of the experiment.

Examples

For examples of how boosted trees are used in machine learning, see the [Azure AI Gallery](#):

- [Demand estimation](#): Uses **Boosted Decision Tree Regression** to predict the number of rentals for a particular time.
- [Twitter sentiment analysis](#): Uses regression to generate a predicted rating.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

TIP

In general, decision trees yield better results when features are somewhat related. If features have a large degree of entropy (that is, they are not related), they share little or no mutual information, and ordering them in a tree will not yield a lot of predictive significance.

Implementation details

The ensemble of trees is produced by computing, at 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.

- For binary classification problem, the output is converted to probability by using some form of calibration.
- For regression problems, the output is the predicted value of the function.
- For ranking problems, the instances are ordered by the output value of the ensemble.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Maximum number of leaves per tree	$>=1$	Integer	20	Specify the maximum number of leaves per tree
Minimum number of samples per leaf node	$>=1$	Integer	10	Specify the minimum number of cases required to form a leaf node
Learning rate	[double.Epsilon;1.0]	Float	0.2	Specify the initial learning rate
Total number of trees constructed	$>=1$	Integer	100	Specify the maximum number of trees that can be created during training
Random number seed	any	Integer		Provide a seed for the random number generator used by the model. Leave blank for default.
Allow unknown categorical levels	any	Boolean	true	If true, create an additional level for each categorical column. Levels in the test dataset not available in the training dataset are mapped to this additional level.

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained regression model

See also

[A-Z Module List](#)

[Regression](#)

Decision Forest Regression

3/10/2021 • 7 minutes to read • [Edit Online](#)

Creates a regression model using the decision forest algorithm

Category: [Initialize Model - Regression](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Decision Forest Regression** module in Azure Machine Learning Studio (classic), to create a regression model based on an ensemble of decision trees.

After you have configured the model, you must train the model using a labeled dataset and the [Train Model](#) module. The trained model can then be used to make predictions. Alternatively, the untrained model can be passed to [Cross-Validate Model](#) for cross-validation against a labeled data set.

How decision forests work in regression tasks

Decision trees are non-parametric models that perform a sequence of simple tests for each instance, traversing a binary tree data structure until a leaf node (decision) is reached.

Decision trees have these 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 and are resilient in the presence of noisy features.

This regression model consists of an ensemble of decision trees. Each tree in a regression decision forest outputs a Gaussian distribution as a 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.

For more information about the theoretical framework for this algorithm and its implementation, see this article: [Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning](#)

How to configure Decision Forest Regression Model

1. Add the **Decision Forest Regression** module to the experiment. You can find the module in Studio (classic) under **Machine Learning**, **Initialize Model**, and **Regression**.
2. Open the module properties, and for **Resampling method**, choose the method used to create the individual trees. You can choose from **Bagging** or **Replicate**.
 - **Bagging:** Bagging is also called *bootstrap aggregating*. Each tree in a regression decision forest

outputs a Gaussian distribution by way of prediction. The aggregation is to find a Gaussian whose first two moments match the moments of the mixture of Gaussians given by combining all Gaussians returned by individual trees.

For more information, see the Wikipedia entry for [Bootstrap aggregating](#).

- **Replicate:** In replication, each tree is trained on exactly the same input data. The determination of which split predicate is used for each tree node remains random and the trees will be diverse.

For more information about the training process with the **Replicate** option, see [Decision Forests for Computer Vision and Medical Image Analysis. Criminisi and J. Shotton. Springer 2013..](#)

3. Specify how you want the model to be trained, by setting the **Create trainer mode** option.

- **Single Parameter**

If you know how you want to configure the model, you can provide a specific set of values as arguments. You might have learned these values by experimentation or received them as guidance.

- **Parameter Range**

If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using a parameter sweep to find the optimal configuration.

[Tune Model Hyperparameters](#) will iterate over all possible combinations of the settings you provided and determine the combination of settings that produces the optimal results.

4. For **Number of decision trees**, indicate the total number of decision trees to create in the ensemble. By creating more decision trees, you can potentially get better coverage, but training time will increase.

TIP

This value also controls the number of trees displayed when visualizing the trained model. If you want to see or print a single tree, you can set the value to 1; however, this means that only one tree will be produced (the tree with the initial set of parameters) and no further iterations will be performed.

5. For **Maximum depth of the decision trees**, type a number to limit the maximum depth of any decision tree. Increasing the depth of the tree might increase precision, at the risk of some overfitting and increased training time.

6. For **Number of random splits per node**, type the number of splits to use when building each node of the tree. A *split* means that features in each level of the tree (node) are randomly divided.

7. For **Minimum number of samples per leaf node**, indicate the minimum number of cases that are required to create any terminal node (leaf) in a tree.

By increasing this value, you increase the threshold for creating new rules. For example, with the default value of 1, even a single case can cause a new rule to be created. If you increase the value to 5, the training data would have to contain at least 5 cases that meet the same conditions.

8. Select the **Allow unknown values for categorical features** option to create a group for unknown values in the training or validation sets.

If you deselect it, the model can accept only the values that are contained in the training data. In the former case, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.

9. Connect a labeled dataset, select a single label column containing no more than two outcomes, and

connect either [Train Model](#) or [Tune Model Hyperparameters](#).

- If you set **Create trainer mode** option to **Single Parameter**, train the model by using the [Train Model](#) module.
- If you set **Create trainer mode** option to **Parameter Range**, train the model by using [Tune Model Hyperparameters](#).

10. Run the experiment.

Results

After training is complete:

- To see the tree that was created on each iteration, right-click the output of the training module, and select **Visualize**.
- To see the rules for each node, click each tree and drill down into the splits.
- To save a snapshot of the trained model, right-click the output of the training module, and select **Save As Trained Model**. This copy of the model is not updated on successive runs of the experiment.

Examples

For examples of regression models, see these sample experiments in the [Cortana Intelligence Gallery](#):

- [Compare Regression Models sample](#): Contrasts several different kinds of regression models.
- [Sentiment analysis sample](#): Uses several different regression models to generate predicted ratings.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

- If you pass a parameter range to [Train Model](#), it will use only the first value in the parameter range list.
- If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and using the default values for the learner.
- If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified will be used throughout the sweep, even if other parameters change across a range of values.

Usage tips

If you have limited data or want to minimize the time spent training the model, try these settings:

Limited training set. If the training set contains a limited number of instances:

- Create the decision forest using a large number of decision trees (for example, more than 20)
- Use the **Bagging** option for resampling
- Specify a large number of random splits per node (for example, more than 1000)

Limited training time. If the training set contains a large number of instances and training time is limited:

- Create the decision forest using fewer decision trees (for example, 5-10)
- Use the **Replicate** option for resampling
- Specify a small number of random splits per node (for example, less than 100)

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Resampling method	any	ResamplingMethod	Bagging	Choose a resampling method
Number of decision trees	>=1	Integer	8	Specify the number of decision trees to create in the ensemble
Maximum depth of the decision trees	>=1	Integer	32	Specify the maximum depth of any decision tree that can be created in the ensemble
Number of random splits per node	>=1	Integer	128	Specify the number of splits generated per node, from which the optimal split is selected
Minimum number of samples per leaf node	>=1	Integer	1	Specify the minimum number of training samples required to generate a leaf node
Allow unknown values for categorical features	any	Boolean	true	Indicate whether unknown values of existing categorical features can be mapped to a new, additional feature

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained regression model

See also

[Regression](#)

[A-Z Module List](#)

Fast Forest Quantile Regression

3/10/2021 • 11 minutes to read • [Edit Online](#)

Creates a quantile regression model

Category: [Machine Learning / Initialize Model / Regression](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Fast Forest Quantile Regression** module in Azure Machine Learning Studio (classic), to create a regression model that can predict values for a specified number of quantiles.

Quantile regression is useful if you want to understand more about the distribution of the predicted value, rather than get a single mean prediction value. This method has many applications, including:

- Predicting prices
- Estimating student performance or applying growth charts to assess child development
- Discovering predictive relationships in cases where there is only a weak relationship between variables

This regression algorithm is a **supervised** learning method, which means it requires a tagged dataset that includes a label column. Because it is a regression algorithm, the label column must contain only numerical values.

More about quantile regression

There are many different types of regression. In the most basic sense, regression means fitting a model to a target expressed as a numeric vector. However, statisticians have been developing increasingly advanced methods for regression.

The simplest definition of *quantile* is a value that divides a set of data into equal-sized groups; thus, the quantile values mark the boundaries between groups. Statistically speaking, quantiles are values taken at regular intervals from the inverse of the cumulative distribution function (CDF) of a random variable.

Whereas linear regression models attempt to predict the value of a numeric variable using a single estimate, the *mean*, sometimes you need to predict the range or entire distribution of the target variable. Techniques such as Bayesian regression and quantile regression have been developed for this purpose.

Quantile regression helps you understand the distribution of the predicted value. Tree-based quantile regression models, such as the one used in this module, have the additional advantage that they can be used to predict non-parametric distributions.

For additional implementation details and resources, see the [Technical Notes](#) section.

How to configure Fast_Forest Quantile Regression

You configure the properties of the regression model using this module, and then train it using one of the

training modules.

Configuration steps differ considerably depending on whether you are providing a fixed set of parameters, or setting up a parameter sweep.

- [To create a quantile regression model using fixed parameters](#)
- [To create a quantile regression model using a parameter sweep](#)

Create a quantile regression model using fixed parameters

Assuming you know how you want to configure the model, you can provide a specific set of values as arguments. When you train the model, use [Train Model](#).

1. Add the **Fast Forest Quantile Regression** module to your experiment in Studio (classic).
2. Set the **Create trainer mode** option to **Single Parameter**.
3. For **Number of Trees**, type the maximum number of trees that can be created in the ensemble. If you create more trees, it generally leads to greater accuracy, but at the cost of longer training time.
4. For **Number of Leaves**, type the maximum number of leaves, or terminal nodes, that can be created in any tree.
5. For **Minimum number of training instances required to form a leaf**, specify the minimum number of examples that are required to create any terminal node (leaf) in a tree.
By increasing this value, you increase the threshold for creating new rules. For example, with the default value of 1, even a single case can cause a new rule to be created. If you increase the value to 5, the training data would have to contain at least 5 cases that meet the same conditions
6. For **Bagging fraction**, specify a number between 0 and 1 that represents the fraction of samples to use when building each group of quantiles. Samples are chosen randomly, with replacement.
7. For **Feature fraction**, type a number between 0 and 1 that indicates the fraction of total features to use when building any particular tree. Features are always chosen randomly.
8. For **Split fraction**, type a number between 0 and 1 that represents the fraction of features to use in each split of the tree. The features used are always chosen randomly.
9. For **Quantile sample count**, type the number of cases to evaluate when estimating the quantiles.
10. For **Quantiles to be estimated**, type a comma-separated list of the quantiles for which you want the model to train and create predictions.

For example, if you want to build a model that estimates for quartiles, you would type `0.25, 0.5, 0.75`.

11. Optionally, type a value for **Random number seed** to seed the random number generator used by the model. The default is 0, meaning a random seed is chosen.

You should provide a value if you need to reproduce results across successive runs on the same data.

12. Select the **Allow unknown categorical levels** option to create a group for unknown values.

If you deselect it, the model can accept only the values that are contained in the training data.

If you select this option, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.

13. Connect a training dataset, select a single label column, and connect [Train Model](#).

14. Run the experiment.

Use a parameter sweep to create a quantile regression model

If you are not sure of the optimal parameters for the model, you can configure a parameter sweep, and provide a range of values as arguments. When you train the model, use the [Tune Model Hyperparameters](#) module.

1. Add the **Fast Forest Quantile Regression** module to your experiment in Studio (classic).
2. Set the **Create trainer mode** option to **Parameter Range**.

A parameter sweep is recommended if you are not sure of the best parameters. By specifying multiple values and using the [Tune Model Hyperparameters](#) module to train the model, you can find the optimal set of parameters for your data.

After choosing a parameter sweep, for each property that is tunable, you can set either a single value, or multiple values. For example, you might decide to fix the number of trees, but randomly change other values that control the way each tree is built.

- If you type a single value, that value is used across all iterations of the sweep, even if other values change.
- Type a comma-separated list of discrete values to use. These values are used in combination with other properties.
- Use the **Range Builder** to define a range of continuous values.

During the training process, the [Tune Model Hyperparameters](#) module iterates over various combinations of the values to build the best model.

3. For **Maximum number of leaves per tree**, type the total number of leaves, or terminal nodes, to allow in each tree.
4. For **Number of trees constructed**, type the number of iterations to perform when constructing the ensemble. By creating more trees, you can potentially get better coverage, at the expense of increased training time.

5. For **Minimum number of sample per leaf node**, indicate how many cases are required to create a leaf node.

By increasing this value, you increase the threshold for creating new rules. For example, with the default value of 1, even a single case can cause a new rule to be created. If you increase the value to 5, the training data would have to contain at least 5 cases that meet the same conditions.

6. In **Range for bagging fraction**, type the fraction of samples to use when building each group of quantiles. Samples are chosen randomly, with replacement.

Each fraction should be a number between 0 and 1. Separate multiple fractions, by using commas.

7. In **Range for feature fraction**, type the fraction of total features to use when building each group of quantiles. Features are chosen randomly.

Each fraction should be a number between 0 and 1; separate multiple fractions by using commas.

8. In **Range for split fraction**, specify some fraction of features to use in each group of quantiles. The actual features used are chosen randomly.

Each fraction should be a number between 0 and 1; separate multiple fractions by using commas.

9. In **Sample count used to estimate the quantiles**, indicate how many samples should be evaluated when estimating the quantiles. If you type a number greater than the number of available samples, all samples are used.

10. In **Required quantile values**, type a comma-separated list of the quantiles on which you want the

model to train. For example, if you want to build a model that estimates quartiles, you would type `0.25, 0.5, 0.75

11. In **Random number seed**, type a value to seed the random number generator used by the model. Use of a seed is useful in order to reproduce duplicate runs.

The default is 0, meaning a random seed is chosen.

12. Select the **Allow unknown values for categorical features** option to create a group for unknown values in the training or validation sets.

If you deselect this option, the model can accept only the values that are contained in the training data.

If you select this option, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.

13. Connect a training dataset, select the label column, and connect the [Tune Model Hyperparameters](#) module.

NOTE

Do not use [Train Model](#). If you configure a parameter range but train using [Train Model](#), it uses only the first value in the parameter range list.

14. Run the experiment.

Results

After training is complete:

- To see the final hyperparameters of the optimized model, right-click the output of [Tune Model Hyperparameters](#) and select **Visualize**.

Examples

For examples of how to use this module, see the [Azure AI Gallery](#):

- [Quantile Regression](#): Demonstrates how to build and interpret a quantile regression model, using the auto price dataset.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

The **Fast Forest Quantile Regression** module in Azure Machine Learning is an implementation of random forest quantile regression using decision trees. Random forests can be helpful to avoid overfitting that can occur with decision trees. A decision tree is a binary tree-like flow chart, where at every interior node, one decides which of the two child nodes to continue to, based on the value of one of the features of the input.

In 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.

In a random forest, an ensemble of trees is created by using bagging to select a subset of random samples and features of the training data, and then fit a decision tree to each subset of data. Unlike the random forest algorithm, which averages out the output of all the trees, **Fast Forest Quantile Regression** keeps all the predicted labels in trees specified by the parameter **Quantile sample count** and outputs the distribution, so

that the user can view the quantile values for the given instance.

Related research

For more information about quantile regression, see these books and articles:

- Quantile Regression Forests. Nicolai Meinshausen

<http://jmlr.org/papers/volume7/meinshausen06a/meinshausen06a.pdf>

- Random forests. Leo Breiman.

<https://rd.springer.com/article/10.1023%2FA%3A1010933404324>

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DESCRIPTION	DEFAULT
Create trainer mode	CreateLearnerMode	List:Single Parameter Parameter Range	Required	Single Parameter	Create advanced learner options
Number of Trees	Integer		mode:Single Parameter	100	Specify the number of trees to be constructed
Number of Leaves	Integer		mode:Single Parameter	20	Specify the maximum number of leaves per tree. The default number is 20
Minimum number of training instances required to form a leaf	Integer		mode:Single Parameter	10	Indicates the minimum number of training instances required to form a leaf
Bagging fraction	Float		mode:Single Parameter	0.7	Specifies the fraction of training data to use for each tree
Feature fraction	Float		mode:Single Parameter	0.7	Specifies the fraction of features (chosen randomly) to use for each tree
Split fraction	Float		mode:Single Parameter	0.7	Specifies the fraction of features (chosen randomly) to use for each split

Name	Type	Range	Optional	Description	Default
Quantile sample count	Integer	Max: 2147483647	mode:Single Parameter	100	Specifies number of instances used in each node to estimate quantiles
Quantiles to be estimated	String		mode:Single Parameter	"0.25;0.5;0.75"	Specifies the quantile to be estimated
Random number seed	Integer		Optional		Provide a seed for the random number generator used by the model. Leave blank for default.
Allow unknown categorical levels	Boolean		Required	true	If true, create an additional level for each categorical column. Levels in the test dataset not available in the training dataset are mapped to this additional level.
Maximum number of leaves per tree	ParameterRange Settings	[16;128]	mode:Parameter Range	16; 32; 64	Specify range for the maximum number of leaves allowed per tree
Number of trees constructed	ParameterRange Settings	[1;256]	mode:Parameter Range	16; 32; 64	Specify the range for the maximum number of trees that can be created during training
Minimum number of samples per leaf node	ParameterRange Settings	[1;10]	mode:Parameter Range	1; 5; 10	Specify the range for the minimum number of cases required to form a leaf
Range for bagging fraction	ParameterRange Settings	[0.25;1.0]	mode:Parameter Range	0.25; 0.5; 0.75	Specifies the range for fraction of training data to use for each tree

NAME	TYPE	RANGE	OPTIONAL	DESCRIPTION	DEFAULT
Range for feature fraction	ParameterRange Settings	[0.25;1.0]	mode:Parameter Range	0.25; 0.5; 0.75	Specifies the range for fraction of features (chosen randomly) to use for each tree
Range for split fraction	ParameterRange Settings	[0.25;1.0]	mode:Parameter Range	0.25; 0.5; 0.75	Specifies the range for fraction of features (chosen randomly) to use for each split
Sample count used to estimate the quantiles	Integer		mode:Parameter Range	100	Sample count used to estimate the quantiles
Required quantile values	String		mode:Parameter Range	"0.25;0.5;0.75"	Required quantile value used during parameter sweep

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained quantile regression model that can be connected to the Train Generic Model or Cross Validate Model modules.

See also

[Regression](#)

Linear Regression

3/10/2021 • 10 minutes to read • [Edit Online](#)

Creates a linear regression model

Category: [Machine Learning](#) / [Initialize Model](#) / [Regression](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Linear Regression** module in Azure Machine Learning Studio (classic), to create a linear regression model for use in an experiment. Linear regression attempts to establish a linear relationship between one or more independent variables and a numeric outcome, or dependent variable.

You use this module to define a linear regression method, and then train a model using a labeled dataset. The trained model can then be used to make predictions. Alternatively, the untrained model can be passed to [Cross-Validate Model](#) for cross-validation against a labeled data set.

More about linear regression

Linear regression is a common statistical method, which has been adopted in machine learning and enhanced with many new methods for fitting the line and measuring error. In the most basic sense, regression refers to prediction of a numeric target. Linear regression is still a good choice when you want a very simple model for a basic predictive task. Linear regression also tends to work well on high-dimensional, sparse data sets lacking complexity.

Azure Machine Learning Studio (classic) supports a variety of regression models, in addition to linear regression. However, the term "regression" can be interpreted loosely, and some types of regression provided in other tools are not supported in Studio (classic).

- The classic regression problem involves a single independent variable and a dependent variable. This is called *simple regression*. This module supports simple regression.
- *Multiple linear regression* involves two or more independent variables that contribute to a single dependent variable. Problems in which multiple inputs are used to predict a single numeric outcome are also called *multivariate linear regression*.

The **Linear Regression** module can solve these problems, as can most of the other regression modules in Studio (classic).

- *Multi-label regression* is the task of predicting multiple dependent variables within a single model. For example, in multi-label logistic regression, a sample can be assigned to multiple different labels. (This is different from the task of predicting multiple levels within a single class variable.)

This type of regression is not supported in Azure Machine Learning. To predict multiple variables, create a separate learner for each output that you wish to predict.

For years statisticians have been developing increasingly advanced methods for regression. This is true even for linear regression. This module supports two methods to measure error and fit the regression line: ordinary least squares method, and gradient descent.

- **Gradient descent** is a method that minimizes the amount of error at each step of the model training process. There are many variations on gradient descent and its optimization for various learning problems has been extensively studied. If you choose this option for **Solution method**, you can set a variety of parameters to control the step size, learning rate, and so forth. This option also supports use of an integrated parameter sweep.
- **Ordinary least squares** is one of the most commonly used techniques in linear regression. For example, least squares is the method that is used in the Analysis Toolpak for Microsoft Excel.

Ordinary least squares refers to the loss function, which computes error as the sum of the square of distance from the actual value to the predicted line, and fits the model by minimizing the squared error. This method assumes a strong linear relationship between the inputs and the dependent variable.

How to configure Linear Regression

This module supports two methods for fitting a regression model, with very different options:

- [Create a regression model using online gradient descent](#)

Gradient descent is a better loss function for models that are more complex, or that have too little training data given the number of variables.

This option also supports a parameter sweep, if you train the model using [Tune Model Hyperparameters](#) to automatically optimize the model parameters.

- [Fit a regression model using ordinary least squares](#)

For small datasets, it is best to select ordinary least squares. This should give very similar results to Excel.

Create a regression model using ordinary least squares

1. Add the **Linear Regression Model** module to your experiment in Studio (classic).

You can find this module in the **Machine Learning** category. Expand **Initialize Model**, expand **Regression**, and then drag the **Linear Regression Model** module to your experiment.

2. In the **Properties** pane, in the **Solution method** dropdown list, select **Ordinary Least Squares**. This option specifies the computation method used to find the regression line.

3. In **L2 regularization weight**, type the value to use as the weight for L2 regularization. We recommend that you use a non-zero value to avoid overfitting.

To learn more about how regularization affects model fitting, see this article: [L1 and L2 Regularization for Machine Learning](#)

4. Select the option, **Include intercept term**, if you want to view the term for the intercept.

Deselect this option if you don't need to review the regression formula.

5. For **Random number seed**, you can optionally type a value to seed the random number generator used by the model.

Using a seed value is useful if you want to maintain the same results across different runs of the same experiment. Otherwise, the default is to use a value from the system clock.

6. Deselect the option, **Allow unknown categorical levels**, if you want missing values to raise an error.

If this option is selected, an additional level is created for each categorical column. Any levels in the test dataset that were not present in the training dataset are mapped to this additional level.

7. Add the [Train Model](#) module to your experiment, and connect a labeled dataset.
8. Run the experiment.

Results for ordinary least squares model

After training is complete:

- To view the model's parameters, right-click the trainer output and select **Visualize**.
- To make predictions, connect the trained model to the [Score Model](#) module, along with a dataset of new values.
- To perform cross-validation against a labeled data set, connect the untrained model to [Cross-Validate Model](#).

Create a regression model using online gradient descent

1. Add the [Linear Regression Model](#) module to your experiment in Studio (classic).

You can find this module in the **Machine Learning** category. Expand **Initialize Model**, expand **Regression**, and drag the [Linear Regression Model](#) module to your experiment
2. In the **Properties** pane, in the **Solution method** dropdown list, choose **Online Gradient Descent** as the computation method used to find the regression line.
3. For **Create trainer mode**, indicate whether you want to train the model with a predefined set of parameters, or if you want to optimize the model by using a parameter sweep.
 - **Single Parameter**: If you know how you want to configure the linear regression network, you can provide a specific set of values as arguments.
 - **Parameter Range**: If you want the algorithm to find the best parameters for you, set **Create trainer mode** option to **Parameter Range**. You can then specify multiple values for the algorithm to try.
4. For **Learning rate**, specify the initial learning rate for the stochastic gradient descent optimizer.
5. For **Number of training epochs**, type a value that indicates how many times the algorithm should iterate through examples. For datasets with a small number of examples, this number should be large to reach convergence.
6. **Normalize features**: If you have already normalized the numeric data used to train the model, you can deselect this option. By default, the module normalizes all numeric inputs to a range between 0 and 1.

NOTE

Remember to apply the same normalization method to new data used for scoring.

7. In **L2 regularization weight**, type the value to use as the weight for L2 regularization. We recommend that you use a non-zero value to avoid overfitting.

To learn more about how regularization affects model fitting, see this article: [L1 and L2 Regularization for Machine Learning](#)
8. Select the option, **Average final hypothesis**, to average the final hypothesis.

In regression models, hypothesis testing means using some statistic to evaluate the probability of the null

hypothesis, which states that there is no linear correlation between a dependent and independent variable. In many regression problems, you must test a hypothesis involving more than one variable.

This option is enabled by default, meaning the algorithm tests a combination of the parameters where two or more parameters are involved.

9. Select the option, **Decrease learning rate**, if you want the learning rate to decrease as iterations progress.
10. For **Random number seed**, you can optionally type a value to seed the random number generator used by the model. Using a seed value is useful if you want to maintain the same results across different runs of the same experiment.
11. Deselect the option, **Allow unknown categorical levels**, if you want missing values to raise an error. When this option is selected, an additional level is created for each categorical column. Any levels in the test dataset not present in the training dataset are mapped to this additional level.
12. Add a labeled dataset and one of the [training modules](#).

If you are not using a parameter sweep, use the [Train Model](#) module.

To have the algorithm find the best parameters for you, train the model using [Tune Model Hyperparameters](#).

NOTE

If you configure the model with specific values using the **Single Parameter** option and then switch to the **Parameter Range** option, the model is trained using the minimum value in the range for each parameter.

Conversely, if you configure specific settings when you create the model but select the **Parameter Range** option, the model is trained using the default values for the learner as the range of values to sweep over.

13. Run the experiment.

Results for online gradient descent

After training is complete:

- To make predictions, connect the trained model to the [Score Model](#) module, together with new input data.
- To perform cross-validation against a labeled data set, connect the untrained model to [Cross-Validate Model](#).

Examples

For examples of regression models, see these sample experiments in the [Azure AI Gallery](#):

- [Compare Regressors](#): Contrasts several different kinds of regression models.
- [Cross Validation for Regression](#): Demonstrates linear regression using ordinary least squares.
- [Twitter sentiment analysis](#): Uses several different regression models to generate predicted ratings.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Usage tips

Many tools support creation of linear regression, ranging from the simple to complex. For example, you can easily perform linear regression in Excel, using the Solver Toolpak, or you can code your own regression algorithm, using R, Python, or C#.

However, because linear regression is a well-established technique that is supported by many different tools, there are many different interpretations and implementations. Not all types of models are supported equally by all tools. There are also some differences in nomenclature to observe.

- Regression methods are often categorized by the number of response variables. For example, multiple linear regression means a model that has multiple variables to predict.
- In Matlab, multivariate regression refers to a model that has multiple response variables.
- In Azure Machine Learning, regression models support a single response variable.
- In the R language, the features provided for linear regression depend on the package you are using. For example, the `glm` package will give you the ability to create a logistic regression model with multiple independent variables. In general, Azure Machine Learning Studio (classic) provides the same functionality as the R `glm` package.

We recommend that you use this module, [Linear Regression](#), for typical regression problems.

In contrast, if you are using multiple variables to predict a class value, we recommend the [Two-Class Logistic Regression](#) or [Multiclass Logistic Regression](#) modules.

If you want to use other linear regression packages that are available for the R language, we recommend that you use the [Execute R Script](#) module and call the `lm` or `glm` packages, which are included in the runtime environment of Azure Machine Learning Studio (classic).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Normalize features	any	Boolean	true	Indicate whether instances should be normalized
Average final hypothesis	any	Boolean	true	Indicate whether the final hypothesis should be averaged
Learning rate	<code>>=double.Epsilon</code>	Float	0.1	Specify the initial learning rate for the stochastic gradient descent optimizer
Number of training epochs	<code>>=0</code>	Integer	10	Specify how many times the algorithm should iterate through examples. For datasets with a small number of examples, this number should be large to reach convergence.
Decrease learning rate	Any	Boolean	true	Indicate whether the learning rate should decrease as iterations progress

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
L2 regularization weight	$>=0.0$	Float	0.001	Specify the weight for L2 regularization. Use a non-zero value to avoid overfitting.
Random number seed	any	Integer		Specify a value to seed the random number generator used by the model. Leave blank for default.
Allow unknown categorical levels	any	Boolean	true	Indicate whether an additional level should be created for each categorical column. Any levels in the test dataset not available in the training dataset are mapped to this additional level.
Include intercept term	Any	Boolean	True	Indicate whether an additional term should be added for the intercept

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained regression model

See also

[Regression](#)

Neural Network Regression

3/10/2021 • 12 minutes to read • [Edit Online](#)

Creates a regression model using a neural network algorithm

Category: [Machine Learning / Initialize Model / Regression](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Neural Network Regression** module in Azure Machine Learning Studio (classic), to create a regression model using a customizable neural network algorithm.

Although neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are easily adapted to regression problems. Any class of statistical models can be termed a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Thus neural network regression is suited to problems where a more traditional regression model cannot fit a solution.

Neural network regression is a supervised learning method, and therefore requires a *tagged dataset*, which includes a label column. Because a regression model predicts a numerical value, the label column must be a numerical data type.

You can train the model by providing the model and the tagged dataset as an input to [Train Model](#) or [Tune Model Hyperparameters](#). The trained model can then be used to predict values for the new input examples.

How to configure Neural Network Regression

Neural networks can be extensively customized. This section describes how to create a model using two methods:

- [Create a neural network model using the default architecture](#)

If you accept the default neural network architecture, use the **Properties** pane to set parameters that control the behavior of the neural network, such as the number of nodes in the hidden layer, learning rate, and normalization.

Start here if you are new to neural networks. The module supports many customizations, as well as model tuning, without deep knowledge of neural networks.

- [Define a custom architecture for a neural network](#)

Use this option if you want to add extra hidden layers, or fully customize the network architecture, its connections, and activation functions.

This option is best if you are already somewhat familiar with neural networks. You use the [Net# language](#) to define the network architecture.

Create a neural network model using the default architecture

1. Add the **Neural Network Regression** module to your experiment in Studio (classic). You can find this module under **Machine Learning**, **Initialize**, in the **Regression** category.

2. Indicate how you want the model to be trained, by setting the **Create trainer mode** option.

- **Single Parameter:** Choose this option if you already know how you want to configure the model.
- **Parameter Range:** Choose this option if you are not sure of the best parameters. Then, specify a range of values and use the [Tune Model Hyperparameters](#) module to iterate over the combinations and find the optimal configuration.

3. In **Hidden layer specification**, select **Fully-connected case**. This option creates a model using the default neural network architecture, which for a neural network regression model, has these attributes:

- The network has exactly one hidden layer.
- The output layer is fully connected to the hidden layer and the hidden layer is fully connected to the input layer.
- The number of nodes in the hidden layer can be set by the user (default value is 100).

Because the number of nodes in the input layer is determined by the number of features in the training data, in a regression model there can be only one node in the output layer.

4. For **Number of hidden nodes**, type the number of hidden nodes. The default is one hidden layer with 100 nodes. (This option is not available if you define a custom architecture using Net#.)

5. For **Learning rate**, type a value that defines the step taken at each iteration, before correction. A larger value for learning rate can cause the model to converge faster, but it can overshoot local minima.

6. For **Number of learning iterations**, specify the maximum number of times the algorithm processes the training cases.

7. For **The initial learning weights diameter**, type a value that determines the node weights at the start of the learning process.

8. For **The momentum**, type a value to apply during learning as a weight on nodes from previous iterations.

9. For **The type of normalizer**, choose one of the following methods to use for feature normalization:

- **Binning normalizer:** Binning creates groups of equal size, and then normalizes every value in each group to be divided by the total number of groups.
- **Gaussian normalizer:** Gaussian normalization rescales the values of each feature to have mean 0 and variance 1. This is done by computing the mean and the variance of each feature, and then, for each instance, subtracting the mean value and dividing by the square root of the variance (the standard deviation).
- **Min-Max normalizer:** Min-max normalization linearly rescales every feature to the [0,1] interval.

Rescaling to the [0,1] interval is done by shifting the values of each feature so that the minimal value is 0, and then dividing by the new maximal value (which is the difference between the original maximal and minimal values).

- **Do not normalize:** No normalization is performed.

10. Select the option, **Shuffle examples**, to change the order of cases between iterations. If you deselect this option, cases are processed in exactly the same order each time you run the experiment.

11. For **Random number seed**, you can optionally type a value to use as the seed. Specifying a seed value is useful when you want to ensure repeatability across runs of the same experiment.

12. Select the option **Allow unknown categorical levels** to create a grouping for unknown values. The model might be less precise on known values but provide better predictions for new (unknown) values.
If you deselect this option, the model can accept only the values contained in the training data.
13. Connect a training dataset and one of the [training modules](#):
 - If you set **Create trainer mode** to **Single Parameter**, use [Train Model](#).
 - If you set **Create trainer mode** to **Parameter Range**, use [Tune Model Hyperparameters](#).

WARNING

If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

14. Run the experiment.

Define a custom architecture

1. Add the [Neural Network Regression](#) module to your experiment.
2. Indicate how you want the model to be trained, by setting the **Create trainer mode** option.
 - **Single Parameter:** Choose this option if you already know how you want to configure the model.
 - **Parameter Range:** Choose this option if you are not sure of the best parameters. Then, specify a range of values and use the [Tune Model Hyperparameters](#) module to iterate over the combinations and find the optimal configuration.
3. In **Hidden layer specification**, select **Custom definition script**. You must choose this option if you want to define a custom neural network architecture by using the **Net#** language.
4. After you select the **Custom definition script** option, the **Neural network definition** text box is displayed. You can paste in Net# script to define a custom architecture for the neural network, including the number of hidden layers, their connections, and advanced options such as specifying the mappings between layers.
5. For **Learning rate**, type a value that defines the step taken at each iteration, before correction. A larger value for learning rate can cause the model to converge faster, but it can overshoot local minima.
6. For **Number of learning iterations**, specify the maximum number of times the algorithm processes the training cases.
7. For **The initial learning weights diameter**, type a value that determines the node weights at the start of the learning process.
8. For **The momentum**, type a value to apply during learning as a weight on nodes from previous iterations.
9. For **The type of normalizer**, choose one of the following methods to use for feature normalization:
 - **Binning normalizer:** Binning creates groups of equal size, and then normalizes every value in each group, by dividing by the total number of groups.
 - **Gaussian normalizer:** Gaussian normalization rescales the values of each feature to have mean

0 and variance 1. This is done by computing the mean and the variance of each feature, and then, for each instance, subtracting the mean value and dividing by the square root of the variance (the standard deviation).

- **Min-Max:** Min-max normalization linearly rescales every feature to the [0,1] interval.

Rescaling to the [0,1] interval is done by shifting the values of each feature so that the minimal value is 0, and then dividing by the new maximal value (which is the difference between the original maximal and minimal values).

- **Do not normalize:** No normalization is performed.

10. Select the option, **Shuffle examples**, to change the order of cases between iterations. If you deselect this option, cases are processed in exactly the same order each time you run the experiment.

11. For **Random number seed**, you can optionally type a value to use as the seed. Specifying a seed value is useful when you want to ensure repeatability across runs of the same experiment.

12. Select the option **Allow unknown categorical levels** to create a grouping for unknown values. Any unknown values in the test data set are mapped to this unknown category. Using this option might make the model slightly less precise on known values but provide better predictions for new (unknown) values.

If you deselect this option, the model can make predictions only for the values contained in the training data.

13. Connect a training dataset and one of the [training modules](#):

- If you set **Create trainer mode** to **Single Parameter**, use [Train Model](#).
- If you set **Create trainer mode** to **Parameter Range**, use [Tune Model Hyperparameters](#).

WARNING

If you pass a parameter range to [Train Model](#), it will use only the first value in the parameter range list.

If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and using the default values for the learner.

If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified will be used throughout the sweep, even if other parameters change across a range of values.

14. Run the experiment.

Results

After training is complete:

- To see a summary of the model's parameters, together with the feature weights learned from training, and other parameters of the neural network, right-click the output of [Train Model](#) or [Tune Model Hyperparameters](#), and select **Visualize**.
- To save a snapshot of the trained model, right-click the **Trained model** output and select **Save As Trained Model**. This model is not updated on successive runs of the same experiment.
- To perform cross-validation against a labeled data set, connect the untrained model to [Cross-Validate Model](#).

Examples

For examples of how this algorithm is used in experiments, see these samples in the [Azure AI Gallery](#):

- [Compare Multiple Regressors](#): Demonstrates the use of several regression algorithms and compares their results.

The experiments provide more help on Net#. The experiments are related and progress from basic to advanced configurations:

- [Deep Neural networks example \(part A\)](#)
- [Deep Neural networks example \(part B\)](#)
- [Deep Neural networks example \(part C\)](#)
- [Deep Neural networks example \(part D\)](#)

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

More about Net#

In Azure Machine Learning Studio (classic), you can customize the architecture of a neural network model by using the Net# language. Customizations supported by the Net# language include:

- Specifying the number of hidden layers and the number of nodes in each layer
- Specifying mappings between layers
- Defining convolutions and weight-sharing bundles
- Choosing the activation function

A neural network model is defined by the structure of its graph, which includes these attributes:

- The number of hidden layers
- The number of nodes in each hidden layer
- How the layers are connected
- Which activation function is used
- Weights on the graph edges

IMPORTANT

The overall structure of the graph, as well as the activation function, can be specified by the user. However, the weights on the edges cannot be specified, and must be learned when training the neural network on the input data.

In general, the network has these defaults:

- The first layer is always the input layer.
- The last layer is always the output layer.
- The number of nodes in the output layer should be equal to the number of classes.

You can define any number of intermediate layers (sometimes called hidden layers, because they are contained within the model, and they are not directly exposed as endpoints).

The Net# reference guide explains the syntax and provides sample network definitions. It explains how you can use Net# to add hidden layers and define the way that the different layers interact with each other.

For example, the following script uses the `auto` keyword, which sets the number of features automatically for input and output layers, and uses the default values for the hidden layer.

```

input Data auto;
hidden Hidden auto from Data all;
output Result auto from Hidden all;

```

For additional script examples, see [Guide to the Net# Neural Networks Specification Language](#).

TIP

Neural networks can be computationally expensive, due to a number of hyperparameters and the introduction of custom network topologies. Although in many cases neural networks produce better results than other algorithms, obtaining such results may involve fair amount of sweeping (iterations) over hyperparameters.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Hidden layer specification	List	Neural Network Topology	Fully-connected case	Specify the architecture of the hidden layer or layers
The initial learning weights diameter	>=double.Epsilon	Float	0.1	Specify the node weights at the start of the learning process
Learning rate	[double.Epsilon;0.01]	Float	0.005	Specify the size of each step in the learning process
The momentum	[0.0;1.0]	Float	0.0	Specify a weight to apply during learning to nodes from previous iterations
Neural network definition	Any	StreamReader		When you select "Custom definition script", type a valid script expression on each line to define the layers, nodes, and behavior of a custom neural network
The type of normalizer	List	Normalization Method	Min-Max normalizer	Select the type of normalization to apply to learning examples
Number of hidden nodes	Any	String	100	Type the number of nodes in the hidden layer. For multiple hidden layers, type a comma-separated list.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Number of learning iterations	$>= 1$	Integer	100	Specify the number of iterations while learning
Shuffle examples	Any	Boolean	true	Select this option to change the order of instances between learning iterations
Random number seed	Any	Integer		Specify a numeric seed to use for random number generation. Leave blank to use the default seed. This parameter is optional
Allow unknown categorical levels	Any	Boolean	true	Indicate whether an additional level should be created for unknown categories. If the test dataset contains categories not present in the training dataset they are mapped to this unknown level.

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained regression model

See also

[Regression](#)

[A-Z Module List](#)

Ordinal Regression

3/10/2021 • 4 minutes to read • [Edit Online](#)

Creates an ordinal regression model

Category: [Machine Learning / Initialize Model / Regression](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Ordinal Regression** module in Azure Machine Learning Studio (classic), to create a regression model that can be used to predict ranked values.

Some examples of ranked values:

- Survey responses that capture user's preferred brands on a 1 to 5 scale
- The order of finishers in a race
- URLs in ranked search results

More about ordinal regression

Ordinal regression is used when the label or target column contains numbers, but the numbers represent a ranking or order rather than a numeric measurement.

Predicting ordinal numbers requires a different algorithm than predicting the values of numbers on a continuous scale, because the numbers assigned to represent rank order do not have intrinsic scale.

For example, to predict students' test scores, you would use a standard regression model, because students' test scores vary on a continuous scale and can be measured. However, to predict their class ranking, you must use an ordinal regression model.

For more information about the research behind this algorithm, see this paper (downloadable PDF):
<https://papers.nips.cc/paper/3125-ordinal-regression-by-extended-binary-classification.pdf>

How to configure Ordinal Regression

This module solves a ranking problem as a series of related classification problems. Therefore, the algorithm creates a series of extended training examples using a binary model for each rank, and trains against that extended set. This operation can be computationally expensive.

1. Add the **Ordinal Regression Model** module to your experiment in Studio (classic). You can find this module under **Machine Learning - Initialize**, in the **Regression** category.
2. Add a module that supports binary classification, and configure the model. There are several two-class modules in the [classification](#) category.
3. Connect the binary classification model as an input to the **Ordinal Regression Model** module.

4. Additional parameters are not required on the **Ordinal Regression Model**; the algorithm has been pre-configured with the most effective parameters for solving a ranking problem.
5. Connect a training dataset and the [Train Model](#) module.
6. In the [Train Model](#) module, select the column that contains the rank values.

The rank values must be numerical values, but they need not be integers or positive numbers, as long as they represent a sequence.

For purposes of processing, the ranks are assumed to have the order 1 to K, where 1 is lowest rank, and K is the highest rank. However, the [Train Model](#) module can work even if the semantics of your scale are reversed.

For example, if in your original survey, 1 was the highest score and 5 is the lowest, it does not affect the processing of the model.

7. Run the experiment.

Results

After training is complete:

- To make predictions, connect the trained model, along with new data, to the [Score Model](#) module.
- To perform cross-validation against a labeled data set, connect the **untrained model** to [Cross-Validate Model](#).

Examples

For examples of how ordinal regression is used in machine learning, see the [Azure AI Gallery](#).

- [Predictive Maintenance - Step C](#): In this sample, **Ordinal Regression** is used to rank values output by a classification model, on the assumption that the value reflects the severity of the failure classification.

Technical notes

The ordinal regression algorithm used in this learner is implemented by extended binary classification, as described by the paper titled *Ordinal Regression by Extended Binary Classification*, by Ling Li and Hsuan-Tien Lin, in NIPS 2006.

Restrictions on input data

You can use any numeric column as the target of an ordinal regression model, but in practice you should use only data that represents some sort of order or ranking.

The intervals between ranks are assumed to be unknown and the size of the interval does not matter to the model; however, the model assumes that the sequence of ranks follows the natural ordering of numbers.

The model itself does not assign any meaning to a particular scale. In other words, you might create one model in which 1 is a good rank and 10 is the worst, and in another model assume that 10 is the desired rank and 1 is the worst.

Ranking algorithm

The training set (X, Y) consists of input vectors x and labels y . The labels represents ranks ranging from 1 to k in sequence: 1, 2, ..., K . The ranks are assumed to be ordered such that 1 is the lowest or the worst rank, and K is the best or highest rank.

The crux of the algorithm lies in modifying the given input features X and labels Y to use extended examples, and then using a binary classifier to solve the ordinal regression problem. The binary classifier is trained to give

a yes/no answer to the question, "Is the rank greater than r?"

For example, for each case in the training set there are $K-1$ extended examples, and the maximum observed rank is K . The extended features are formed by appending the i^{th} row of a $K-1 \times K-1$ identity matrix to the input features for all i . The labels are given +1 for the first $r-1$ rows if its rank is r and -1 to the rest.

Sample calculations

To illustrate how it works, let x^1 be the training feature whose rank is 3, where the maximum observed rank is 5. The extended examples corresponding to this feature are as follows:

CASE	TEST	RESULTING LABEL
X11000	Is rank greater than 1?	Yes; therefore +1
X10100	Is rank greater than 2?	Yes; therefore +1
X10010	Is rank greater than 3?	No; therefore no additional feature
X10001	Is rank greater than 4?	No; therefore no additional feature

Expected inputs

NAME	TYPE	DESCRIPTION
Untrained binary classification model	ILearner interface	An untrained binary classification model

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained ordinal regression model

See also

[Regression](#)

Poisson Regression

3/10/2021 • 6 minutes to read • [Edit Online](#)

Creates a regression model that assumes data has a Poisson distribution

Category: [Machine Learning / Initialize Model / Regression](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Poisson Regression** module in Azure Machine Learning Studio (classic) to create a Poisson regression model.

Poisson regression is intended for use in regression models that are used to predict numeric values, typically counts. Therefore, you should use this module to create your regression model only if the values you are trying to predict fit the following conditions:

- The response variable has a [Poisson distribution](#).
- Counts cannot be negative. The method will fail outright if you attempt to use it with negative labels.
- A Poisson distribution is a discrete distribution; therefore, it is not meaningful to use this method with non-whole numbers.

TIP

If your target isn't a count, Poisson regression is probably not an appropriate method. Try one of the other modules in this category. For help picking a regression method, see the [Azure machine Learning algorithm cheat sheet](#).

After you have set up the regression method, you must train the model using a dataset containing examples of the value you want to predict. The trained model can then be used to make predictions.

More about Poisson regression

Poisson regression is a special type of regression analysis that is typically used to model counts. For example, Poisson regression would be useful in these scenarios:

- Modeling the number of colds associated with airplane flights
- Estimating the number of emergency service calls during an event
- Projecting the number of customer inquiries subsequent to a promotion
- Creating contingency tables

Because the response variable has a Poisson distribution, the model makes different assumptions about the data and its probability distribution than, say, least-squares regression. Therefore, Poisson models should be

interpreted differently from other regression models.

How to configure Poisson Regression

1. Add the **Poisson Regression** module to your experiment in Studio (classic).

You can find this module under **Machine Learning - Initialize**, in the **Regression** category.

2. Add a dataset that contains training data of the correct type.

We recommend that you use [Normalize Data](#) to normalize the input dataset before using it to train the regressor.

3. In the **Properties** pane of the **Poisson Regression** module, specify how you want the model to be trained, by setting the **Create trainer mode** option.

- **Single Parameter:** If you know how you want to configure the model, provide a specific set of values as arguments.
- **Parameter Range.** If you are not sure of the best parameters, do a parameter sweep using the [Tune Model Hyperparameters](#) module. The trainer iterates over multiple values you specify to find the optimal configuration.

4. **Optimization tolerance:** Type a value that defines the tolerance interval during optimization. The lower the value, the slower and more accurate the fitting.

5. **L1 regularization weight** and **L2 regularization weight:** Type values to use for L1 and L2 regularization. *Regularization* adds constraints to the algorithm regarding aspects of the model that are independent of the training data. Regularization is commonly used to avoid overfitting.

- L1 regularization is useful if the goal is to have a model that is as sparse as possible.

L1 regularization is done by subtracting the L1 weight of the weight vector from the loss expression that the learner is trying to minimize. The L1 norm is a good approximation to the L0 norm, which is the number of non-zero coordinates.

- L2 regularization prevents any single coordinate in the weight vector from growing too much in magnitude. L2 regularization is useful if the goal is to have a model with small overall weights.

In this module, you can apply a combination of L1 and L2 regularizations. By combining L1 and L2 regularization, you can impose a penalty on the magnitude of the parameter values. The learner tries to minimize the penalty, in a tradeoff with minimizing the loss.

For a good discussion of L1 and L2 regularization, see [L1 and L2 Regularization for Machine Learning](#).

6. **Memory size for L-BFGS:** Specify the amount of memory to reserve for model fitting and optimization.

L-BFGS is a specific method for optimization, based on the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. The method uses a limited amount of memory (L) to compute the next step direction.

By changing this parameter, you can affect the number of past positions and gradients that are stored for computation of the next step.

7. Connect the training dataset and the untrained model to one of the training modules:

- If you set **Create trainer mode** to **Single Parameter**, use the [Train Model](#) module.
- If you set **Create trainer mode** to **Parameter Range**, use the [Tune Model Hyperparameters](#) module.

WARNING

- If you pass a parameter range to [Train Model](#), it uses only the first value in the parameter range list.
- If you pass a single set of parameter values to the [Tune Model Hyperparameters](#) module, when it expects a range of settings for each parameter, it ignores the values and uses the default values for the learner.
- If you select the **Parameter Range** option and enter a single value for any parameter, that single value you specified is used throughout the sweep, even if other parameters change across a range of values.

8. Run the experiment to train the model.

Examples

For examples of how Poisson regression is used in machine learning, see the [Azure AI Gallery](#).

- [Sample 6: Train, Test, Evaluate for Regression: Auto Imports Dataset](#): This experiment compares the outcomes of two algorithms: **Poisson Regression** and **Decision Forest Regression**.
- [Preventive Maintenance](#): An extended walkthrough that uses **Poisson Regression** to assess the severity of failures predicted by a decision forest model.

Technical notes

Poisson regression is used to model count data, assuming that the label has a Poisson distribution. For example, you might use it to predict the number of calls to a customer support center on a particular day.

For this algorithm, it is assumed that an unknown function, denoted Y, has a Poisson distribution. The Poisson distribution is defined as follows:

Given the instance $x = (x_0, \dots, x_{d-1})$, for every $k=0,1, \dots$, the module computes the probability that the value of the instance is k .

Given the set of training examples, the algorithm tries to find the optimal values for $\theta_0, \dots, \theta_{D-1}$ by trying to maximize the log likelihood of the parameters. The likelihood of the parameters $\theta_0, \dots, \theta_{D-1}$ is the probability that the training data was sampled from a distribution with these parameters.

The log probability can be viewed as $\log p(y = y_i)$

The prediction function outputs the expected value of that parameterized Poisson distribution, specifically:
 $f_w, b(x) = E[Y|x] = e^{w^T x + b}$.

For more information, see the entry for [Poisson regression](#) in Wikipedia.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Optimization tolerance	$>= \text{double.Epsilon}$	Float	0.0000001	Specify a tolerance value for optimization convergence. The lower the value, the slower and more accurate the fitting.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
L1 regularization weight	$>=0.0$	Float	1.0	Specify the L1 regularization weight. Use a non-zero value to avoid overfitting the model.
L2 regularization weight	$>=0.0$	Float	1.0	Specify the L2 regularization weight. Use a non-zero value to avoid overfitting the model.
Memory size for L-BFGS	$>=1$	Integer	20	Indicate how much memory (in MB) to use for the L-BFGS optimizer. With less memory, training is faster but less accurate the training.
Random number seed	any	Integer		Type a value to seed the random number generator used by the model. Leave blank for default.
Allow unknown categorical levels	any	Boolean	true	Indicate whether an additional level should be created for each categorical column. Any levels in the test dataset not available in the training dataset are mapped to this additional level.

Outputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	An untrained regression model

See also

[Regression](#)

[A-Z Module List](#)

Machine Learning - Score

3/10/2021 • 4 minutes to read • [Edit Online](#)

This section lists the modules provided in Azure Machine Learning Studio (classic) for *scoring*.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Scoring is also called prediction, and is the process of generating values based on a trained machine learning model, given some new input data. The values or scores that are created can represent predictions of future values, but they might also represent a likely category or outcome. The meaning of the score depends on the type of data you provide, and the type of model that you created.

Create and use models in Machine Learning Studio (classic)

The typical workflow for machine learning includes these phases:

- Choosing a suitable algorithm, and setting initial options.
- Training the model on compatible data.
- Creating predictions using new data, based on the patterns in the model.
- Evaluating the model to determine if the predictions are accurate, how much error there is, and if there is any overfitting.

Machine Learning Studio (classic) supports a flexible, customizable framework for machine learning. Each task in this process is performed by a specific type of module, which can be modified, added, or removed, without breaking the rest of your experiment.

The modules in this section include tools for scoring. In this phase of machine learning, you apply a trained model to new data, to generate predictions. You can either send those predictions to an application that consumes machine learning results, or use the results of scoring to evaluate the accuracy and usefulness of the model.

More about scoring

Scoring is widely used in machine learning to mean the process of generating new values, given a model and some new input. The generic term "score" is used, rather than "prediction," because the scoring process can generate so many different types of values:

- A list of recommended items and a similarity score.
- Numeric values, for time series models and regression models.
- A probability value, indicating the likelihood that a new input belongs to some existing category.
- The name of a category or cluster to which a new item is most similar.
- A predicted class or outcome, for classification models.

NOTE

You might have also heard the word *score* used to mean a weight or value assigned as a result of data analysis. However, in Machine Learning Studio (classic), scoring usually denotes the process of generating predicted values from new data.

When you add one of these modules in your experiment, you must attach an already trained machine learning model, and some new data. When you run the experiment or the selected module, the scoring module ingests the new data, computes scores based on the model, and returns the scores in a table.

Data used for scoring

The new data that you provide as input generally needs to have the same columns that were used to train the model, minus the label, or outcome column.

Columns that are used solely as identifiers are usually excluded when training a model, and thus should be excluded when scoring as well. However, identifiers such as primary keys can easily be re-combined with the scoring dataset later, by using the [Add Columns](#) module. This module works without you having to specify a join key, as long as the dataset size has not changed.

Before you perform scoring on your dataset, always check for missing values and nulls. When data used as input for scoring has missing values, the missing values are used as inputs. Because nulls are propagated, the result is usually a missing value.

List of scoring modules

Machine Learning Studio (classic) provides many different scoring modules. You select one depending on the type of model you are using, or the type of scoring task you are performing:

- [Apply Transformation](#): Applies a well-specified data transformation to a dataset.

Use this module to apply a saved process to a set of data.

- [Assign Data to Clusters](#): Assigns data to clusters by using an existing trained clustering model.

Use this module if you want to cluster new data based on an existing K-Means clustering model.

This module replaces the Assign to Clusters (deprecated) module, which has been deprecated but is still available for use in existing experiments.

- [Score Matchbox Recommender](#): Scores predictions for a dataset by using the Matchbox recommender.

Use this module if you want to generate recommendations, find related items or users, or predict ratings.

- [Score Model](#): Scores predictions for a trained classification or regression model.

Use this module for all other regression and classification models, as well as some anomaly detection models.

Related tasks

- Special scoring modules are provided for Vowpal Wabbit. See [Text Analytics](#).
- You can score for special classes of images on pretrained models by using the [OpenCV Library](#).
- The [Time Series Anomaly Detection](#) module generates scores that represent potential deviations from a trend.

Examples

These examples in the [Azure AI Gallery](#) demonstrate the process of scoring, from basic to advanced scenarios:

- [Binary Classification for Direct Marketing](#): Demonstrates the basic workflow for scoring, in a scenario where the predicted value is customer response to a marketing campaign.
- [Predict Book Reviews](#): Scoring on text data. Uses a logistic regression model.
- [Learning with Counts](#): Shows you how to use count-based featurization for making predictions.
- [No-code batch scoring with Logic Apps and Azure Machine Learning](#): Illustrates the end-to-end process of training and scoring, automated with the Logic Apps feature of Azure App Service.

The following articles provide real-world examples of how you can use a machine learning model for scoring:

- [Develop a predictive solution](#)
- [Survival analysis](#)
- [Anomaly detection](#)
- [Lexicon-based sentiment analysis](#)

See also

- [Module Categories and Descriptions](#)
- [Train](#)
- [Score](#)
- [Evaluate](#)

Apply Transformation

3/10/2021 • 3 minutes to read • [Edit Online](#)

Applies a well-specified data transformation to a dataset

Category: [Machine Learning / Score](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Apply Transformation](#) module in Azure Machine Learning Studio (classic), to modify an input dataset based on a previously computed transformation.

For example, if you used z-scores to normalize your training data by using the [Normalize Data](#) module, you would want to use the z-score value that was computed for training during the scoring phase as well. In Azure Machine Learning Studio (classic), you can do this easily by saving the normalization method as a transform, and then using [Apply Transformation](#) to apply the z-score to the input data before scoring.

Azure Machine Learning Studio (classic) provides support for creating and then applying many different kinds of custom transformations. For example, you might want to save and then re-use transformations that do the following:

- Remove or replace missing values, using [Clean Missing Data](#)
- Bin, scale, and normalize data, using [Normalize Data](#) or [Group Data into Bins](#)
- Create a set of compact features by calculating joint probability distribution for a dataset, using the [Learning with Counts](#) modules.

How to use Apply Transformation

1. Add the [Apply Transformation](#) module to your experiment. You can find thi module under **Machine Learning**, in the **Score** category.
2. Locate an existing transformation to use as an input.

If the transformation was created earlier in the experiment (for example, as part of a cleaning or data scaling operation) typically the [ITransform interface](#) object is available on the module's right-hand output. Connect that output to the left-hand input of [Apply Transformation](#).

Previously saved transformations can be found in the **Transforms** group in the left navigation pane.

TIP

If you design a transformation for an experiment but do not explicitly save it, the transformation is available in the workspace as long as your session is open. If you close the session but do not save the transformation, you can re-run the experiment to generate the [ITransform interface](#) object.

3. Connect the dataset that you want to transform. The dataset should have exactly the same schema (number of columns, column names, data types) as the dataset for which the transformation was first designed.
4. No other parameters need to be set; all customization is done when defining the transformation.
5. To apply a transformation to the new dataset, run the experiment.

Examples

To see how this module is used in machine learning, see the [Azure AI Gallery](#):

- [Online Fraud Detection](#): This sample demonstrates how to use **Apply Transformation** with [Clean Missing Data](#), to ensure that missing values are handled the same in all datasets.
- [Predictive Maintenance](#): Demonstrates how to use **Apply Transformation** with [Normalize Data](#).
- [Learning with Counts](#): Uses **Apply Transformation** to reuse a count table.

Technical notes

The **Apply Transformation** module can take as input the output of any module that creates an [ITransform interface](#). These modules include:

- [Clean Missing Data](#)
- [Normalize Data](#)
- [Group Data into Bins](#)
- [Learning with Counts](#)

TIP

You can also save and re-use filters designed for digital signal processing. However, filters use the [IFilter interface](#) interface, rather than [ITransform interface](#).

Expected inputs

NAME	TYPE	DESCRIPTION
Transformation	ITransform interface	A unary data transformation
Dataset	Data Table	Dataset to be transformed

Outputs

NAME	TYPE	DESCRIPTION
Transformed dataset	Data Table	Transformed dataset

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Filter](#)

[Apply SQL Transformation](#)

[Clean Missing Data](#)

[Normalize Data](#)

[A-Z Module List](#)

[Group Data into Bins](#)

Assign Data to Clusters

3/10/2021 • 3 minutes to read • [Edit Online](#)

Assigns data to clusters using an existing trained clustering model

Category: [Score](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Assign Data to Clusters](#) module in Azure Machine Learning Studio (classic), to generate predictions using a clustering model that was trained using the K-Means clustering algorithm.

The module returns a dataset that contains the probable assignments for each new data point. It also creates a PCA (Principal Component Analysis) graph to help you visualize the dimensionality of the clusters.

WARNING

This module replaces the Assign to Clusters (deprecated) module, which is available only for support of older experiments.

How to use Assign Data to Clusters

1. In Azure Machine Learning Studio (classic), locate a previously trained clustering model. You can create and train a clustering model by using either of these methods:

- Configure the K-means algorithm using the [K-Means Clustering](#) module, and then train the model using a dataset and the [Train Clustering Model](#) module.
- Configure a range of options for the K-means algorithm using [K-Means Clustering](#) and then train the model using the [Sweep Clustering](#) module.

You can also add an existing trained clustering model from the [Saved Models](#) group in your workspace.

2. Attach the trained model to the left input port of [Assign Data to Clusters](#).

3. Attach a new dataset as input. In this dataset, labels are optional. Generally, clustering is an unsupervised learning method so it is not expected that you will know categories in advance.

However, the input columns must be the same as the columns that were used in training the clustering model, or an error occurs.

TIP

To reduce the number of columns output from cluster predictions, use [Select Columns in Dataset](#), and select a subset of the columns.

- Leave the option **Check for Append or Uncheck for Result Only** selected if you want the results to contain the full input dataset, together with a column indicating the results (cluster assignments).

If you deselect this option, you get back just the results. This might be useful when creating predictions as part of a web service.

- Run the experiment.

Results

The [Assign Data to Clusters](#) module returns two types of results on the **Results dataset** output:

- To see the separation of clusters in the model, click the output of the module and select **Visualize**

This command displays a Principal Component Analysis (PCA) graph that maps the collection of values in each cluster to two component axes.

- The first component axis is the combined set of features that captures the most variance in the model. It is plotted on the x-axis (**Principal Component 1**).
- The next component axis represents some combined set of features that is orthogonal to the first component and that adds the next most information to the chart. It is plotted on the y-axis (**Principal Component 2**).

From the graph, you can see the separation between the clusters, and how the clusters are distributed along the axes that represent the principal components.

- To view the table of results for each case in the input data, attach the [Convert to Dataset](#) module, and visualize the results in Studio (classic).

This dataset contains the *cluster assignments* for each case, and a distance metric that gives you some indication of how close this particular case is to the center of the cluster.

OUTPUT COLUMN NAME	DESCRIPTION
Assignments	A 0-based index that indicates which cluster the data point was assigned to.
DistancesToClusterCenter no. <i>n</i>	For each data point, this value indicates the distance from the data point to the center of the assigned cluster, and the distance to other clusters. The metric used to calculate distance is determined when you configure the K-means clustering model.

Expected inputs

NAME	TYPE	DESCRIPTION
Trained model	ICluster interface	Trained clustering model
Dataset	Data Table	Input data source

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Append or Result Only			Required	TRUE	Indicate whether the output dataset should contain the input dataset as well as the results, or the results only
Specify parameter sweeping mode	Sweep Methods	List:Entire grid Random sweep	Required	Random sweep	Sweep entire grid on parameter space, or sweep with using a limited number of sample runs

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Input dataset appended by data column of assignments or assignments column only

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.

See also

[K-Means Clustering Score](#)

Score Matchbox Recommender

3/10/2021 • 21 minutes to read • [Edit Online](#)

Scores predictions for a dataset using the Matchbox recommender

Category: [Machine Learning / Score](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Score Matchbox Recommender** module in Azure Machine Learning Studio (classic), to create predictions based on a trained recommendation model, based on the Matchbox algorithm from Microsoft Research.

The Matchbox recommender can generate four different kinds of predictions:

- [Predict ratings for a given user and item](#)
- [Recommend items to a given user](#)
- [Find users related to a given user](#)
- [Find items related to a given item](#)

When creating the latter three kinds of predictions, you can operate in either *production mode* or *evaluation mode*.

- **Production mode** considers all users or items, and is typically used in a web service.

You can create scores for new users, not just users seen during training. For more information, see [this section](#).

- **Evaluation mode** operates on a reduced set of users or items that can be evaluated, and is typically used during experimentation.

TIP

Learn everything you need to know about the end-to-end experience of building a recommendation system in this tutorial from the .NET development team. Includes sample code and discussion of how to call Azure Machine Learning from an application.

[Building recommendation engine for .NET applications using Azure Machine Learning](#)

More about the Matchbox recommender

The goal of creating a recommendation system is to recommend one or more "items" to "users" of the system. Examples of an item could be a movie, restaurant, book, or song. A user could be a person, group of persons, or other entity with item preferences.

There are two principal approaches to recommender systems. The first is the **content-based** approach, which makes use of features for both users and items. Users may be described by properties such as age and gender, and items may be described by properties such as author and manufacturer. Typical examples of content-based recommendation systems can be found on social matchmaking sites. The second approach is **collaborative filtering**, which uses only identifiers of the users and the items and obtains implicit information about these entities from a (sparse) matrix of ratings given by the users to the items. We can learn about a user from the items they have rated and from other users who have rated the same items.

The Matchbox recommender combines collaborative filtering with a content-based approach. It is therefore considered a **hybrid recommender**. When a user is relatively new to the system, predictions are improved by making use of the feature information about the user, thus addressing the well-known "cold-start" problem. However, once there are a sufficient number of ratings from a particular user, it is possible to make fully personalized predictions for them based on their specific ratings rather than on their features alone. Hence, there is a smooth transition from content-based recommendations to recommendations based on collaborative filtering. Even when user or item features are not available, Matchbox still works in its collaborative filtering mode.

More details on the Matchbox recommender and its underlying probabilistic algorithm can be found in the relevant research paper:

- [Matchbox: Large Scale Bayesian Recommendations](#)

Examples

For examples of how to create scores from a recommendation engine, see the [Azure AI Gallery](#).

- [Recommender: Movie recommendation](#): This sample demonstrates item recommendation, where the items are movies, and also demonstrates rating prediction.
- [Recommender: Restaurant ratings](#): This sample demonstrates item recommendations using both item features and user features.
- [Recommendations Everywhere](#): This blog post provides a high-level introduction to recommender systems with lots of visual aids.

How to configure Score Matchbox Recommender

This module supports different types of recommendations, each with different requirements. Click the link for the type of data you have and the type of recommendation you want to create.

- [Predict ratings](#)
- [Recommend items](#)
- [Find related users](#)
- [Find related items](#)

Predict ratings

When you predict ratings, the model calculates how a given user will react to a particular item, given the training data. Therefore, the input data for scoring must provide both a user and the item to rate.

1. Add a trained recommendation model to your experiment, and connect it to **Trained Matchbox recommender**. You must create the model by using [Train Matchbox Recommender](#).
2. **Recommender prediction kind**: Select **Rating Prediction**. No further parameters are required.
3. Add the data for which you wish to make predictions, and connect it to **Dataset to score**.

To predict ratings, the input dataset must contain user-item pairs.

The dataset can contain an optional third column of ratings for the user-item pair in the first and second columns, but the third column will be ignored during prediction.

4. (Optional). If you have a dataset of user features, connect it to **User features**.

The dataset of user features should contain the user identifier in the first column. The remaining columns should contain values that characterize the users, such as their gender, preferences, location, etc.

Features of users who have rated items are ignored by **Score Matchbox Recommender**, because they have already been learned during training. Therefore, filter your dataset in advance to include only *cold-start users*, or users who have not rated any items.

WARNING

If the model was trained without using user features, you cannot introduce user features during scoring.

5. If you have a dataset of item features, you can connect it to **Item features**.

The item features dataset must contain an item identifier in the first column. The remaining columns should contain values that characterize the items.

Features of rated items are ignored by **Score Matchbox Recommender** as they have already been learned during training. Therefore, restrict your scoring dataset to *cold-start items*, or items that have not been rated by any users.

WARNING

If the model was trained without using item features, you cannot introduce item features during scoring.

6. Use the optional fifth input port, named **Training Dataset**, to remove items that have already been rated from the prediction results.

To apply this filter, connect the original training dataset to the input port.

7. Run the experiment.

Results for rating predictions

The output dataset contains three columns, containing the user, the item, and the predicted rating for each input user and item.

Additionally, the following changes are applied during scoring:

- Missing values in a user or item feature columns are automatically replaced with the **mode** of its non-missing training set values.
- All user and item features are rescaled by the corresponding maximum absolute values seen in training.
- No translation is applied to the feature values, to maintain their sparsity.
- String-valued features are converted into a set of binary-valued indicator features.

Recommend

To recommend items for users, you provide a list of users and items as input. From this data, the model uses its knowledge about existing items and users to generate a list of items with probable appeal to each user. You can customize the number of recommendations returned, and set a threshold for the number of previous recommendations that are required in order to generate a recommendation.

1. Add a trained recommendation model to your experiment, and connect it to **Trained Matchbox recommender**. You must create the model by using [Train Matchbox Recommender](#).
2. To recommend items for a given list of users, set **Recommender prediction kind** to **Item Recommendation**.
3. **Recommended item selection:** Indicate whether you are using the scoring module in production or for model evaluation, by choosing one of these values:
 - **From Rated Items (for model evaluation):** Select this option if you are developing or testing a model. This option enables **evaluation mode**, and the module makes recommendations only from those items in the input dataset that have been rated.
 - **From All Items:** Select this option if you are setting up an experiment to use in a Web service or production. This option enables **production mode**, and the module makes recommendations from all items seen during training.
4. Add the dataset for which you want to make predictions, and connect it to **Dataset to score**.
 - If you choose the option, **From All Items**, the input dataset should consist of one and only one column, containing the identifiers of users for which to make recommendations.
If the dataset contains more than one column, an error is raised. Use the [Select Columns in Dataset](#) module to remove extra columns from the input dataset.
 - If you choose the option, **From Rated Items (for model evaluation)**, the input dataset should consist of **user-item pairs**. The first column should contain the **user** identifier. The second column should contain the corresponding **item** identifiers.
The dataset can include a third column of user-item ratings, but this column is ignored.
5. (Optional). If you have a dataset of **user features**, connect it to **User features**.
The first column in the user features dataset should contain the user identifier. The remaining columns should contain values that characterize the user, such as their gender, preferences, location, etc.
Features of users who have rated items are ignored by **Score Matchbox Recommender**, because these features have already been learned during training. Therefore, you can filter your dataset in advance to include only *cold-start users*, or users who have not rated any items.

WARNING
If the model was trained without using user features, you cannot use apply features during scoring.
6. (Optional) If you have a dataset of **item features**, you can connect it to **Item features**.
The first column in the item features dataset must contain the item identifier. The remaining columns should contain values that characterize the items.
Features of rated items are ignored by **Score Matchbox Recommender**, because these features have already been learned during training. Therefore, you can restrict your scoring dataset to *cold-start items*, or items that have not been rated by any users.

WARNING
If the model was trained without using item features, do not use item features when scoring.
7. **Maximum number of items to recommend to a user:** Type the number of items to return for each

user. By default, 5 items are recommended.

8. **Minimum size of the recommendation pool per user:** Type a value that indicates how many prior recommendations are required. By default, this parameter is set to 2, meaning the item must have been recommended by at least two other users.

This option should be used only if you are scoring in evaluation mode. The option is not available if you select **From All Items**.

9. Run the experiment.

Results of item recommendation

The scored dataset returned by **Score Matchbox Recommender** lists the recommended items for each user.

- The first column contains the user identifiers.
- A number of additional columns are generated, depending on the value you set for **Maximum number of items to recommend to a user**. Each column contains a recommended item (by identifier). The recommendations are ordered by user-item affinity, with the item with highest affinity put in column, **Item 1**.

WARNING

This scored dataset cannot be evaluated using the [Evaluate Recommender](#) module.

Find related users

The option to find related users is useful if you are recommending "people like you", or if you are creating a pool of similar users on which to base other types of predictions.

1. Add a trained recommendation model to your experiment, and connect it to **Trained Matchbox recommender**. You must create the model by using [Train Matchbox Recommender](#).
2. **Recommender prediction kind:** Select **Related Users**.
3. **Related user selection:** Indicate how you will be using the model for scoring, and specify the pool of users on which to base the scores as follows:
 - **From All Users:** Select this option if you are setting up an experiment to use in a Web service or production, or if you need to make predictions for new users. This option enables **production mode**, and the module bases its recommendation only on users seen during training.
 - **From Users That Rated Items (for model evaluation):** Select this option if you are developing or testing a model. This option enables **evaluation mode**, and the model bases its recommendations on the users in the test set who have rated some common items.
4. Connect a dataset that contains the users for which to generate predictions. The format for this dataset depends on whether you are using the scoring module in production mode or evaluation mode.
 - Production mode, using **From All Items**

The dataset to score must consist of **users** for which you wish to find related users. The first and only column should contain the user identifiers. If other columns are included, an error is raised. Use the [Select Columns in Dataset](#) module to remove unnecessary columns.

- Evaluation mode, using **From Rated Items (for model evaluation)**

The dataset to score should consist of 2-3 columns, containing user-item pairs. The first column should contain user identifiers. The second column should contain item identifiers. The dataset can include a third column of ratings (by the user in column 1 for the item in column 2), but the ratings column will be ignored.

5. **Maximum number of related users to find for a user:** Type a number that indicates the maximum number of predictions you want for each user. The default is 5, meaning that at most five related users can be returned, but in some cases there might be fewer than 5.

6. In evaluation mode (**From Users That Rated Items**), configure these additional parameters:

- **Minimum number of items that the query user and the related user must have rated in common:** This value sets a threshold for recommendations. The number that you type represents the minimum number of items that your target user and the potential related user must have rated. The default value is 2, meaning that, at minimum, two items must have been rated by both users.
- **Minimum size of the related user pool for a single user:** This value controls the minimum number of similar users needed to create a recommendation. By default, the value is 2, meaning that if you have as few as two users who are related by virtue of rating the same items, you can consider them related and generate a recommendation.

7. (Optional). If you have a dataset of user features, connect it to **User features**.

The first column in the user features dataset should contain the user identifier. The remaining columns should contain values that characterize the user, such as gender, preferences, location, etc.

Features of users who have rated items are ignored by **Score Matchbox Recommender** as these features have already been learned during training. Therefore, filter your dataset in advance to include only *cold-start users*, or users who have not rated any items.

WARNING

If the model was trained without using user features, you cannot apply user features during scoring.

8. (Optional) If you have a dataset of item features, connect it to **Item features**.

The first column in the item features dataset must contain the item identifier. The remaining columns should contain values that characterize the items.

Features of rated items are ignored by **Score Matchbox Recommender** as these features have already been learned during training. Therefore, you can restrict your scoring dataset to *cold-start items*, or items which have not been rated by any users.

WARNING

If the model was trained without using item features, do not use item features when scoring.

9. Run the experiment.

Results for related users

The scored dataset returned by **Score Matchbox Recommender** lists the users who are related to each users in the input dataset.

For each user specified in the input dataset, the result dataset contains a set of related users.

- The first column contains the identifier of the target user (the user provided as input).
- Additional columns are generated containing the identifiers of related users. The number of additional columns depends on the value you set in the option, **Maximum number of related users to find for a user**.

Related users are ordered by the strength of the relation to the target user, with the most strongly related user in the column, **Related User 1**.

Find related items

By predicting related items, you can generate recommendations for users based on items that have already been rated.

1. Add a trained recommendation model to your experiment, and connect it to **Trained Matchbox recommender**. You must create the model by using [Train Matchbox Recommender](#).
2. **Recommender prediction kind:** Select **Related Items**.
3. Connect a dataset that contains the users for which to generate predictions. The format for this dataset depends on whether you are using the scoring module in production mode or evaluation mode.
 - Production mode, using **From All Items**

The dataset to score must consist of items for which you wish to find related users.

The first and only column should contain the item identifiers. If other columns are included, an error is raised. Use the [Select Columns in Dataset](#) module to remove unnecessary columns.

- Evaluation mode, using **From Rated Items (for model evaluation)**

The dataset to score should consist of 2-3 columns, containing user-item pairs. The first column should contain user identifiers. The second column should contain item identifiers.

The dataset can include a third column of ratings (by the user in column 1 for the item in column 2), but the ratings column are ignored.

4. **Maximum number of related items to find for an item >**: Type a number that indicates the maximum number of predictions you want for each item.

The default is 5, meaning that at most five related items can be returned, but there might be fewer than 5.

5. If you are using evaluation mode (**From Users That Rated Items**), configure these additional parameters:

- **Minimum number of items that the query item and the related item must have been rated by in common:** This value sets a threshold for recommendations. The number that you type represents the minimum number of items that have been rated by the target user and some related user. The default value is 2, meaning that, at minimum, two items must have been rated by the target user and the related user.
- **Minimum size of the related item pool for a single item:** This value controls the minimum number of similar items needed to create a recommendation. By default, the value is 2, meaning that, if you have as few as two items that are related by virtue of having been rated by the same users, you can consider them related and generate a recommendation.

6. (Optional). If you have a dataset of user features, connect it to **User features**.

The first column in the user features dataset should contain the user identifier. The remaining columns should contain values that characterize the user, such as their gender, preferences, location, etc.

Features of users who have rated items are ignored by **Score Matchbox Recommender**, because these features have already been learned during training. Therefore, you can filter your dataset in advance to include only *cold-start users*, or users who have not rated any items.

WARNING

If the model was trained without using user features, you cannot apply user features during scoring.

7. (Optional) If you have a dataset of item features, you can connect it to **Item features**.

The first column in the item features dataset must contain the item identifier. The remaining columns should contain values that characterize the item.

Features of rated items are ignored by **Score Matchbox Recommender**, because these features have already been learned during training. Therefore, you can restrict your scoring dataset to *cold-start items*, or items which have not been rated by any users.

WARNING

If the model was trained without using item features, do not use item features when scoring.

8. (Optional) In a predictive experiment, you can use a fifth input port, named **Training Dataset**, to remove existing users that were included in the model training data from the prediction results.

To apply this filter, connect the original training dataset to the input port.

9. Run the experiment.

Results for related items

The scored dataset returned by **Score Matchbox Recommender** lists the related items for each item in the input dataset.

- The first column contains the identifier of the target item (the item provided as input).
- Additional columns are generated containing the identifiers of related items. The number of additional columns depends on the value you set in the option, **Maximum number of related items to find for an item**.

The related items are ordered by the strength of the relation to the target item, with the most strongly related item in the column, **Related Item 1**.

Technical notes

This section contains answers to some common questions about using the recommender to create predictions.

Cold-start users and recommendations

Typically, to create recommendations, the **Score Matchbox Recommender** module requires the same inputs that you used when training the model, including a user ID. That is because the algorithm needs to know if it has learned something about this user during training.

However, for new users, you might not have a user ID, only some user features such as age, gender, and so forth.

You can still create recommendations for users who are new to your system, by handling them as *cold-start users*. For such users, the recommendation algorithm does not use past history or previous ratings, only user features.

For purposes of prediction, a cold-start user is defined as a user with an ID that has not been used for training. To ensure that IDs do not match IDs used in training, you can create new identifiers. For example, you might generate random IDs within a specified range, or allocate a series of IDs in advance for cold-start users.

However, if you do not have any collaborative filtering data, such as a vector of user features, you are better off

using a classification or regression learner.

Production use of the Matchbox recommender

If you have experimented with the Matchbox recommender and then move the model to production, be aware of these key differences when using the recommender in evaluation mode and in production mode:

- Evaluation, by definition, requires predictions that can be verified against the *ground truth* in a test set. Therefore, when you evaluate the recommender, it must predict only items that have been rated in the test set. This necessarily restricts the possible values that are predicted.

However, when you operationalize the model, you typically change the prediction mode to make recommendations based on all possible items, in order to get the best predictions. For many of these predictions, there is no corresponding ground truth, so the accuracy of the recommendation cannot be verified in the same way as during experimentation.

- If you do not provide a user ID in production, and provide only a feature vector, you might get as response a list of all recommendations for all possible users. Be sure to provide a user ID.

To limit the number of recommendations that are returned, you can also specify the maximum number of items returned per user.

- It is not possible to generate predictions only for items that have not previously been rated. This is by design.

The reason is that, in order to recommend only the items that have not been rated, the recommender would need to store the entire training data set with the model, which would increase your use of storage.

If you want to recommend only items that have not been seen by your user, you can request more items to recommend, and then manually filter out the rated ones.

Continuous update of the recommender

Online updating (or continuous training) of a recommendation model is not currently supported in Azure Machine Learning. If you want to capture user responses to recommendations and use those for improving the model, we suggest retraining the complete model periodically. Incremental training is not possible, but you can apply a sliding window to the training data to ensure that data volume is minimized while using the most recent data.

Expected inputs

NAME	TYPE	DESCRIPTION
Trained Matchbox recommender	ILearner	Trained Matchbox recommender
Dataset to score	Data Table	Dataset to score
User features	Data Table	Dataset containing features that describe users This data is optional
Item features	Data Table	Dataset containing features that describe items This data is optional

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Recommender prediction kind	List	Prediction kind	Item Recommendation	Specify the type of prediction the recommender should output
Recommended item selection	List	Item selection	From Rated Items (for model evaluation)	Select the set of items to make recommendations from
Related user selection	List	User selection	From Users That Rated Items (for model evaluation)	Select the set of users to use when finding related items
Related item selection	List	[Item selection	From Rated Items (for model evaluation)	Select the set of items to use when finding related items

Outputs

NAME	TYPE	DESCRIPTION
Scored dataset	Data Table	Scored dataset

Exceptions

EXCEPTION	DESCRIPTION
Error 0022	Exception occurs if number of selected columns in input dataset does not equal to the expected number.
Error 0036	Exception occurs if multiple feature vectors were provided for a given user or item.
Error 0013	Exception occurs if passed to module learner has invalid type.
Error 0035	Exception occurs if no features were provided for a given user or item.
Error 0053	Exception occurs in the case when there are no user features or items for Matchbox recommendations.
Error 0003	Exception occurs if one or more of inputs are null or empty.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Train Matchbox Recommender](#)

[Evaluate Recommender](#)

[Score](#)

Score Model

3/10/2021 • 2 minutes to read • [Edit Online](#)

Scores predictions for a trained classification or regression model

Category: [Machine Learning / Score](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Score Model](#) module in Azure Machine Learning Studio (classic), to generate predictions using a trained classification or regression model.

How to use Score Model

1. Add the **Score Model** module to your experiment in Studio (classic).
2. Attach a trained model and a dataset containing new input data.

The data should be in a format compatible with the type of trained model you are using. The schema of the input dataset should also generally match the schema of the data used to train the model.

3. Run the experiment.

Results

After you have generated a set of scores using [Score Model](#):

- To generate a set of metrics used for evaluating the model's accuracy (performance), you can connect the scored dataset to [Evaluate Model](#),
- Right-click the module and select **Visualize** to see a sample of the results.
- Save the results to a dataset.

The score, or predicted value, can be in many different formats, depending on the model and your input data:

- For classification models, [Score Model](#) outputs a predicted value for the class, as well as the probability of the predicted value.
- For regression models, [Score Model](#) generates just the predicted numeric value.
- For image classification models, the score might be the class of object in the image, or a Boolean indicating whether a particular feature was found.

Publish scores as a web service

A common use of scoring is to return the output as part of a predictive web service. For more information, see this tutorial on how to create a web service based on an experiment in Azure ML Studio (classic):

- [Publish a web service from Azure ML Studio \(classic\)](#)

Examples

For examples of how [Score Model](#) is used in an experimental workflow, see the [Azure AI Gallery](#):

- [Compare Binary Classification Models](#)
- [Compare Multiclass Classification Models](#)
- [Compare Multiple Regression Models](#)

Technical notes

Models not supported by Score Model

If you are using one of the following special types of model, you might need to use one of these custom scoring modules:

- Score a clustering model: Use [Assign Data to Clusters](#).
- Create recommendations or generate data for evaluating a recommender: Use [Score Matchbox Recommender](#)

Usage tips

If the data that you are scoring contains missing values, in many cases no score will be generated for the entire row.

The following machine learning models require that data have no missing values. When using the following machine learning models, review the data before passing it to [Score Model](#), and use [Clean Missing Data](#) to amend the missing values in input columns.

- [Two-Class Logistic Regression](#)
- [Two-Class Support Vector Machine](#)

Expected inputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	Trained predictive model
Dataset	Data Table	Input test dataset

Outputs

NAME	TYPE	DESCRIPTION
Scored dataset	Data Table	Dataset with obtained scores

Exceptions

EXCEPTION	DESCRIPTION
Error 0032	Exception occurs if argument is not a number.
Error 0033	Exception occurs if argument is Infinity.
Error 0003	Exception occurs if one or more of inputs are null or empty.

EXCEPTION	DESCRIPTION
Error 0013	Exception occurs if the learner that is passed to the module is an invalid type.

See also

[Evaluate](#)

[Train Model](#)

[Score Matchbox Recommender](#)

Machine Learning - Train

3/10/2021 • 4 minutes to read • [Edit Online](#)

This article describes the modules provided in Azure Machine Learning Studio (classic) for training a machine learning model. *Training* is the process of analyzing input data by using the parameters of a predefined model. From this analysis, the model learns the patterns, and saves them in the form of a trained model.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

This article also describes the overall process in Machine Learning Studio (classic) for model creation, training, evaluation, and scoring.

Create and use machine learning models

The typical workflow for machine learning includes these phases:

- Choosing a suitable algorithm, and setting initial options.
- Training the model on compatible data.
- Creating predictions by using new data, based on the patterns in the model.
- Evaluating the model to determine if the predictions are accurate, how much error there is, and if there is any overfitting.

Machine Learning Studio (classic) supports a flexible, customizable framework for machine learning. Each task in this process is performed by a specific type of module, which can be modified, added, or removed, without breaking the rest of your experiment.

The modules in this category support training for different types of models. During training, the data is analyzed by the machine learning algorithm. This algorithm analyzes the distribution and type of the data, compiles statistics, and creates patterns that can be used later for prediction.

More about model training

When Machine Learning is training a model, rows with missing values are skipped. Therefore, if you want to fix the values manually, use imputation, or specify a different method for handling missing values, use the [Clean Missing Data](#) module before training on the dataset.

We recommend that you use the [Edit Metadata](#) module to fix any other issues with the data. You might need to mark the label column, change data types, or correct column names.

For other common data cleanup tasks, such as normalization, sampling, binning, and scaling, see the [Data Transformation](#) category.

Choose the right trainer

The method that you use to train a model depends on the type of model you are creating, and the type of data that the model requires. For example, Machine Learning provides modules specifically for training anomaly detection models, recommendation models, and more.

Check the [List of training modules](#) to determine which one is correct for your scenario.

If you are not sure of the best parameters to use when training a model, use one of the modules provided for parameter sweeping and validation:

- [Tune Model Hyperparameters](#) can perform a parameter sweep on almost all classification and regression models. It trains multiple models, and then returns the best model.
- The [Sweep Clustering](#) module supports model tuning during the training process, and is intended for use only with clustering models. You can specify a range of centroids, and train on data while automatically detecting the best parameters.
- The [Cross-Validate Model](#) module is also useful for model optimization, but does not return a trained model. Instead, it provides metrics that you can use to determine the best model.

Retrain models

If you need to retrain a production model, you can re-run the experiment at any time.

You can also automate the retraining process by using web services. For a walkthrough, see [Retraining and updating Azure Machine Learning models with Azure Data Factory](#).

Use pretrained models

Machine Learning includes some models that are pretrained, such as the [Pretrained Cascade Image Classification](#) module. You can use these models for scoring without additional data input.

Also, some modules (such as [Time Series Anomaly Detection](#)) do not generate a trained model in the iLearner format. But they do take training data and create a model internally, which can then be used to make predictions. To use these, you just configure the parameters and provide data.

Save a snapshot of a trained model

If you want to save or export the model, right-click the training module, and select **Save as Trained Model**. The model is exported to the iLearner format and saved in your workspace, under **Trained Models**. Trained models can be re-used in other experiments, or connected to other modules for scoring.

You can also use the [Load Trained Model](#) module in an experiment to retrieve a stored model.

List of modules

The **Train** category includes these modules:

- [Sweep Clustering](#): Performs a parameter sweep on a clustering model to determine the optimum parameter settings, and trains the best model.
- [Train Anomaly Detection Model](#): Trains an anomaly detector model and labels data from a training set.
- [Train Clustering Model](#): Trains a clustering model and assigns data from the training set to clusters.
- [Train Matchbox Recommender](#): Trains a Bayesian recommender by using the Matchbox algorithm.
- [Train Model](#): Trains a classification or regression model from a training set.
- [Tune Model Hyperparameters](#): Performs a parameter sweep on a regression or classification model to determine the optimum parameter settings, and trains the best model.

Related tasks

Some modules are not in this category, because they require a special format or are customized for a specific task:

- [Train Anomaly Detection Model](#)
- [Train Vowpal Wabbit Version 7-4](#)

- [Train Vowpal Wabbit Version 7-10](#)
- [Train Vowpal Wabbit Version 8](#)
- [Latent Dirichlet Allocation](#)

See also

- [Evaluate](#)
- [Initialize Model](#)

Sweep Clustering

3/10/2021 • 12 minutes to read • [Edit Online](#)

Performs a parameter sweep to determine the optimum settings for a clustering model

Category: [Machine Learning / Train](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Sweep Clustering](#) module in Azure Machine Learning Studio (classic), to train a model using a parameter sweep. A parameter sweep is a way of finding the best hyperparameters for a model, given a set of data.

The [Sweep Clustering](#) module is designed specifically for clustering models. You provide a clustering model as input, together with a dataset. The module iterates over a set of parameters you specify, building and testing models with different parameters, until it finds the model with the best set of clusters. It automatically computes the best configuration, and then trains a model using that configuration.

It also returns a set of metrics describing the models that were tested, and a set of cluster assignments based on the best model.

How to configure Sweep Clustering

1. Add the [Sweep Clustering](#) module to your experiment in Studio (classic). You can find this module under [Machine Learning](#), in the [Train](#) category.
2. Add the [K-Means Clustering](#) module and your training dataset to the experiment, and connect them both to the [Sweep Clustering](#) module.
3. Configure the [K-Means Clustering](#) module to use a parameter sweep as follows:
 - a. Set **Create trainer mode** to **Parameter Range**.
 - b. Use the **Range Builder** (or manually type multiple values) for each parameter to set the range of values to iterate over.
 - c. **Initialization for sweep:** Specify how the K-means algorithm should find the initial cluster centroids. Multiple algorithms are provided for randomly initializing and then testing centroids.

If your training dataset contains a label column, even with partial values, you can use those values for centroids. Use the **Assign Label Mode** option to indicate how the label values are used.

TIP

Your label column must be marked as such in advance. If you get an error, try using [Edit Metadata](#) to identify the column containing labels.

- d. **Number of seeds to sweep:** Indicate how many different random starting seeds to try when doing the parameter sweep.
 - e. Choose the metric to use when measuring cluster similarity. For more information, see the [K-Means Clustering](#) topic.
 - f. **Iterations:** Specify the total number of iterations that the K-means algorithm should perform. These iterations are used to optimize the selection of the cluster centroids.
 - g. If you are using a label column to initialize the sweep, use the **Assign Label Mode** option to specify how the values in the label column should be handled.
 - **Fill missing values:** If your label column contains some missing values, use this option to impute categories based on the cluster the data point is assigned to.
 - **Overwrite from closest to center:** Generates label values for all data points assigned to a cluster, using the label of the point that is closest to the center of the cluster.
 - **Ignore label column:** Select this option if you don't want to perform either of the above operations.
4. In the [Sweep Clustering](#) module, use the option, **For Metric for measuring clustering result**, to specify the mathematical method to use when estimating the fit of the trained clustering model:
 - **Simplified Silhouette:** This metric captures the tightness of data points within each cluster. It is computed as a combination of the similarity of each row with its cluster and its similarity with next closest cluster. If the cluster has only 1 row, the prorated distance to next closest centroid is calculated instead, to avoid getting 0 as the result. "Simplified" refers to the fact that the distance to cluster centroid is used as a simple similarity measure. In general, a higher score is better. The average value over the dataset indicates how well the data has been clustered. If there are too many or too few clusters, some clusters will have lower silhouette values than the rest. For more information, see [this Wikipedia article](#).
 - **Davies-Bouldin:** This metric aims to identify the smallest set of clusters with the least scatter. Because the metric is defined as a ratio of scatter within each cluster over cluster separation, a lower value means that clustering is better. The best clustering model minimizes this metric. To calculate the Davies-Bouldin metric, the average row to centroid distance is computed per cluster. For each pair of clusters, the sum of those averages is divided by the distance between centroids. Maximal value over all other clusters is selected for each cluster and averaged over all clusters. For more information, see [this Wikipedia article](#).
 - **Dunn:** This metric aims to identify the smallest set of most compact clusters. Generally, a higher value for this metric indicates better clustering. To calculate the Dunn metric, the minimal centroid-to-centroid distance is divided by the maximal distance of each data point to its cluster center. For more information, see [this Wikipedia article](#).
 - **Average deviation:** This metric is computed by taking the average distance from each data point to its cluster center. The value decreases as the number of centroids increases; therefore, it is not useful when sweeping to find the number of centroids. This metric is recommended for use when you are choosing the best centroid initialization seed.
 5. **Specify parameter sweeping mode:** Select an option that defines the combinations of values that are

used when training, and how they are chosen:

- **Entire grid**: All values within the given range are tried and evaluated. This option is usually more computationally expensive.
 - **Random sweep**: Use this option to limit the number of runs. The clustering model is built and evaluated using a combination of values chosen randomly from the allowed range of parameter values.
6. **Maximum number of runs on random sweep**: Set this option if you choose the **Random sweep** option. Type a value to limit the maximum number of iterations when testing sets of randomly chosen parameters.

WARNING

The **Iterations** parameters of the **K-Means Clustering** module has a different purpose and is not affected by this setting: it limits the number of passes over the data made to improve clusters, by minimizing the average distance from each data point to its cluster centroids. In contrast, the iterations defined by the **Sweep Clustering** module parameter are performed in order to try different random centroid initializations. This minimization problem is known to be NP-hard; therefore, trying several random seeds could produce better results.

If you select a random sweep, use the **Random seed** option to specify the initial random seed values, on which to begin creating the centroids. One advantage of using a parameter sweep to create a clustering model is that you can easily test multiple seed values to mitigate the known sensitivity of clustering models to the initial seed value.

7. Click **Column Set**, and choose the columns to use when building the clusters. By default, all feature columns are used when building and testing the clustering model.

You can include a label column, if present in your dataset. If a label is present, you can use it to guide selection of centroids, use the label as a feature, or ignore the label. Set these options for label handling the **Kmeans Clustering** module as described in Step 3 above.

8. **Check for Append or Uncheck for Result Only**: Use this option to control which columns are returned in the results.

By default, the module returns the original columns of the training dataset together with the results. If you deselect this option, only the cluster assignments are returned.

9. Add the [Assign Data to Clusters](#) module to your experiment.
10. Connect the output labeled **Best Trained Model** to the **Trained Model** input of [Assign Data to Clusters](#).
11. Add the dataset intended for evaluation, and connect it to the **Dataset** port of the [Assign Data to Clusters](#) module.
12. Add the [Evaluate Model](#) module, and connect it to [Assign Data to Clusters](#). Optionally, you can connect an evaluation dataset.
13. Run the experiment.

Results

The [Sweep Clustering](#) module outputs three different results:

- **Best Trained Model**. A trained model that you can use for scoring and evaluation. Right-click and select **Save as Trained Model** to capture the optimized clustering model and use it for scoring.
- **Results dataset**. A set of cluster assignments, based on the optimized model.

COLUMN NAME	DESCRIPTION
Assignments	This value indicates the cluster to which each data point has been assigned. The clusters in the trained model are labeled with 0-based indexes.
DistancesToClusterCenter no.1 DistancesToClusterCenter no.n	<p>This value indicates how close the data point is to the center of each cluster.</p> <p>A column is created for each cluster created in the optimized model.</p> <p>You can constrain the number of clusters by using the Number of centroids option.</p>

By default, you can return the columns from the training dataset along with the results, to make it easier to review and interpret the cluster assignments.

- **Sweep results.** A dataset containing the following evaluation metrics for the clusters:

COLUMN NAME	DESCRIPTION
Cluster metric	A value that indicates the average cluster quality for that run. The runs are ordered by the best score.
Number of centroids	The number of clusters that were created in this particular iteration of the sweep
Index of run	An identifier for each iteration

TIP

The values returned for the cluster metric should be interpreted differently, depending on which metric you chose when you set up the sweep. For the default metric, **Simplified silhouette**, a higher score is better. For **Davies-Bouldin**, a lower score is better.

Examples

To see examples a parameter sweep with K-means clustering, see the [Azure AI Gallery](#):

- [Clustering sweep using the Diabetes dataset](#)

Technical notes

This section contains tips and implementation details.

Optimizing clustering models

The quality and accuracy of clustering models can be strongly affected by the choice of initial parameters, such as the number of centroids and the seed value used to initialize cluster. To mitigate this sensitivity to the initial parameters, the [Sweep Clustering](#) module helps you find the best combination of parameters. You specify a range of parameters to test, and the module automatically builds and tests multiple models, and finally selects the optimum number of clusters.

To create a parameter sweep, you must also configure the [K-Means Clustering](#) module to use a parameter sweep. You can specify that the sweep iterate over all possible combinations of parameters, or use a random

combination of parameters. You can also choose one of several standard metrics for measuring the accuracy of the centroids during the iterative model building and testing process. After the specified number of iterations has completed, the module selects the best number of clusters, based on the selected metric, and outputs reports that you can use to assess the results.

Usage tips

- In some cases, you might already know how many clusters you expect to find. For example, your data might have class labels that could be used in guiding the selection of the centroids. In that case, you can configure the [K-Means Clustering](#) module to use the label column to guide selection of the initial centroids.
- If you know some of the expected clusters but are not sure how many clusters are optimal, set the number of centroids to a number that is greater than the number of known label values. The [Sweep Clustering](#) module creates clusters for the known data points and then determines the optimal number of extra clusters for the remaining data points.

Handling missing values in the label column

There are several ways to handle missing values in your label column. For example, suppose that you have an image classification task and only some of the images have been labeled.

You can use the label column to guide selection of the centroids but specify that any missing labels be filled in by using the cluster assignments. In other words, existing label values are not changed, but missing labels are filled in.

Alternatively, for all data points assigned to a cluster, you can overwrite even the existing labels, using a single label that best represents the cluster. To understand how this option be helpful, imagine that you are using image data with very detailed labels, such as different dog breeds. Using this option, you could replace all the detailed labels with a single category label, "dog".

Seed values in the log

The log file generated by the [Train Clustering Model](#) module appears to indicate that the same seed is used for all iterations of the K-means clustering algorithm, regardless of the seed that was provided in the **Random Seed** property.

In fact, the implementation uses the user-supplied seed to generate a sequence of random numbers that are different for every run. Thus, only one seed is needed to create all randomly generated numbers.

The intent of the log is indicate which seed the module uses when the user does not specify a seed in the **Properties** pane.

Expected inputs

NAME	TYPE	DESCRIPTION
Untrained model	ICluster interface	Untrained clustering model
Dataset	Data Table	Input data source

Module parameters

Name	Type	Values	Optional	Default	Description
Metric for measuring clustering result	Cluster Metric	Simplified Silhouette, Davies-Bouldin, Dunn, Average Deviation	Required	Simplified Silhouette	Select the metric used for evaluating regression models
Specify parameter sweeping mode	Sweep Methods	Entire grid or Random sweep	Required	Random sweep	Sweep entire grid on parameter space, or sweep with using a limited number of sample runs
Column Set	ColumnSelection		Required		Column selection pattern
Maximum number of runs on random sweep	Integer	[1;10000]	Available only when SweepingMode is set to Random sweep	5	Set maximum number of runs to execute when using random sweep
Random seed	Integer		Available only when SweepingMode is set to Random sweep	0	Provide a value to seed the random number generator for random sweep
Check for Append or Uncheck for Result Only	Boolean		Required	True	Select to indicate that output dataset must contain input dataset with assignments column appended. Deselect to indicate that only assignments column should be output.

Outputs

Name	Type	Description
Best Trained model	ICluster interface	Trained clustering model
Results dataset	Data Table	Input dataset appended by data column of assignments or assignments column only
Sweep results	Data Table	Resulting metric log for cluster sweep runs

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[K-Means Clustering](#)

[Assign Data to Clusters](#)

[Machine Learning / Train](#)

[Machine Learning / Initialize Model / Clustering](#)

Train Anomaly Detection Model

3/10/2021 • 2 minutes to read • [Edit Online](#)

Trains an anomaly detection model on a training set

Category: [Machine Learning / Train](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Train Anomaly Detection Model** module in Azure Machine Learning to create a trained anomaly detection model.

The module takes as input a set of model parameters for anomaly detection model, such as that produced by the [One-Class Support Vector Machine](#) module, and an unlabeled dataset. It returns a trained anomaly detection model, together with a set of labels for the training data.

For more information about the anomaly detection algorithms provided in Azure Machine Learning, see these topics:

- [One-Class Support Vector Machine](#)
- [PCA-Based Anomaly Detection](#)

How to configure Train Anomaly Detection Model

1. Add the **Train Anomaly Detection Model** module to your experiment in Studio (classic). You can find the module under **Machine Learning**, in the **Train** category.
2. Connect one of the modules designed for anomaly detection, such as [PCA-Based Anomaly Detection](#) or [One-Class Support Vector Machine](#).
Other types of models are not supported; on running the experiment you will get the error: All models must have the same learner type.
3. Configure the anomaly detection module by choosing the label column and setting other parameters specific to the algorithm.
4. Attach a training dataset to the right-hand input of **Train Anomaly Detection Model**.
5. Run the experiment.

Results

After training is complete:

- To view the model's parameters, right-click the module and select **Visualize**.
- To create predictions, use [Score Model](#) with new input data.

- To save a snapshot of the trained model, right-click the **Trained Model** output, and select **Save As**.

Examples

For an example of how anomaly detection is implemented in Azure Machine Learning, see the [Azure AI Gallery](#):

- [On-line Fraud Detection](#): Provides a detailed walkthrough of an anomaly detection scenario, including how to engineer features and interpret the results of an algorithm.
- [Anomaly Detection: Credit Risk](#): Illustrates how to use the [One-Class Support Vector Machine](#) and [PCA-Based Anomaly Detection](#) modules for fraud detection.

Expected inputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	Untrained anomaly detection model
Dataset	Data Table	Input data source

Outputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	Trained anomaly detection model

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Train](#)
[Anomaly Detection](#)

Train Clustering Model

3/10/2021 • 3 minutes to read • [Edit Online](#)

Trains a clustering model and assigns data from the training set to clusters

Category: [Machine Learning / Train](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Train Clustering Model** module in Azure Machine Learning Studio (classic), to train a clustering model.

The module takes an untrained clustering model that you have already configured using the [K-Means Clustering](#) module, and trains the model using a labeled or unlabeled data set. The module creates both a trained model that you can use for prediction, and a set of cluster assignments for each case in the training data.

NOTE

A clustering model cannot be trained using the [Train Model](#) module, which is the generic module for creating machine learning models. That is because [Train Model](#) works only with supervised learning algorithms. K-means and other clustering algorithms allow unsupervised learning, meaning that the algorithm can learn from unlabeled data.

How to use Train Clustering Model

1. Add the **Train Clustering Model** module to your experiment in Studio (classic). You can find the module under [Machine Learning Modules](#), in the [Train](#) category.
2. Add the [K-Means Clustering](#) module, or another custom module that creates a compatible clustering model, and set the parameters of the clustering model.
3. Attach a training dataset to the right-hand input of **Train Clustering Model**.
4. In **Column Set**, select the columns from the dataset to use in building clusters. Be sure to select columns that make good features: for example, avoid using IDs or other columns that have unique values, or columns that have all the same values.
If a label is available, you can either use it as a feature, or leave it out.
5. Select the option, **Check for Append or Uncheck for Result Only**, if you want to output the training data together with the new cluster label.
If you deselect this option, only the cluster assignments are output.
6. Run the experiment, or click the **Train Clustering Model** module and select **Run Selected**.

Results

After training has completed:

- To view the cluster and their separation in a graph, right-click the **Results** dataset output and select **Visualize**.

The graph represents the principal components of the cluster, rather than the actual values. See [Principal Component Analysis](#) for more information.

- To view the values in the dataset, add an instance of the [Convert to Dataset](#) module, and connect it to the **Results dataset** output. Run the [Convert to Dataset](#) module to get a copy of the data that you can view or download.
- To save the trained model for later re-use, right-click the module, select **Trained model**, and click **Save As Trained Model**.
- To generate scores from the model, use [Assign Data to Clusters](#).

Examples

For an example of how clustering is used in machine learning, see the [Azure AI Gallery](#):

- [Clustering: Find similar Companies](#): Demonstrates how to use clustering on attributes derived from unstructured text.
- [Clustering: Color quantization](#): Demonstrates how to use clustering to find related colors and reduce the number of bits used in images.
- [Clustering: Group iris data](#): Provides a simple example of clustering based on the iris dataset.

Expected inputs

NAME	TYPE	DESCRIPTION
Untrained model	ICluster interface	Untrained clustering model
Dataset	Data Table	Input data source

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Column Set	any	ColumnSelection		Column selection pattern
Check for Append or Uncheck for Result Only	any	Boolean	true	Whether output dataset must contain input dataset appended by assignments column (Checked) or assignments column only (Unchecked)

Outputs

NAME	TYPE	DESCRIPTION
Trained model	ICluster interface	Trained clustering model
Results dataset	Data Table	Input dataset appended by data column of assignments or assignments column only

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[A-Z Module List](#)

[Train](#)

[Assign Data to Clusters](#)

[K-Means Clustering](#)

Train Matchbox Recommender

3/10/2021 • 9 minutes to read • [Edit Online](#)

Trains a Bayesian recommender using the Matchbox algorithm

Category: [Machine Learning / Train](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Train Matchbox Recommender** module in Azure Machine Learning Studio (classic), to train a recommendation model.

The recommendation algorithm in Azure Machine Learning is based on the **Matchbox** model, developed by [Microsoft Research](#). To download a paper that describes the algorithm in detail, click this link on the [Microsoft Research site](#).

The **Train Matchbox Recommender** module reads a dataset of user-item-rating triples and, optionally, some user and item features. It returns a trained Matchbox recommender. You can then use the trained model to generate recommendations, find related users, or find related items, by using the [Score Matchbox Recommender](#) module.

TIP

Learn everything you need to know about the end-to-end experience of building a recommendation system in this tutorial from the .NET development team. Includes sample code and discussion of how to call Azure Machine Learning from an application.

[Building recommendation engine for .NET applications using Azure Machine Learning](#)

More about recommendation models and the Matchbox recommender

The main aim of a recommendation system is to recommend one or more *items* to *users* of the system. Examples of an item could be a movie, restaurant, book, or song. A user could be a person, group of persons, or other entity with item preferences.

There are two principal approaches to recommender systems.

- The first is the **content-based** approach, which makes use of features for both users and items. Users may be described by properties such as age and gender, and items may be described by properties such as author and manufacturer. Typical examples of content-based recommendation systems can be found on social matchmaking sites.
- The second approach is **collaborative filtering**, which uses only identifiers of the users and the items and obtains implicit information about these entities from a (sparse) matrix of ratings given by the users to the

items. We can learn about a user from the items they have rated and from other users who have rated the same items.

The Matchbox recommender combines these approaches, using collaborative filtering with a content-based approach. It is therefore considered a **hybrid recommender**.

How this works: When a user is relatively new to the system, predictions are improved by making use of the feature information about the user, thus addressing the well-known "cold-start" problem. However, once you have collected a sufficient number of ratings from a particular user, it is possible to make fully personalized predictions for them based on their specific ratings rather than on their features alone. Hence, there is a smooth transition from content-based recommendations to recommendations based on collaborative filtering. Even if user or item features are not available, Matchbox will still work in its collaborative filtering mode.

More details on the Matchbox recommender and its underlying probabilistic algorithm can be found in the relevant research paper: [Matchbox: Large Scale Bayesian Recommendations](#). In addition, the [Machine Learning Blog](#) has an article titled [Recommendations Everywhere](#) that provides a high-level introduction to recommendation algorithms.

How to configure Train Matchbox Recommender

- [Prepare the training data](#)
- [Train the model](#)

Prepare data

Before trying to use the module, it is essential that your data be in the format expected by the recommendation model. A training data set of **user-item-rating triples** is required, but you can also include user features and item features (if available), in separate datasets.

To divide source data into training and testing datasets, use the **Recommender Split** option in the [Split Data](#) module.

Required dataset of user-item-ratings

It is very important that the input data used for training contain the right type of data in the correct format:

- The first column must contain user identifiers.
- The second column must contain item identifiers.
- The third column contains the rating for the user-item pair. Rating values must be either numeric or categorical.

During training, the rating values cannot all be the same. Moreover, if numeric, the difference between the minimum and the maximum rating values must be less than 100, and ideally not greater than 20.

The **Restaurant ratings** dataset in Azure Machine Learning Studio (classic) (click **Saved Datasets** and then **Samples**) demonstrates the expected format:

USERID	PLACEID	RATING
U1077	135085	2
U1077	135038	2

From this sample, you can see that a single user has rated two separate restaurants.

User features dataset (optional)

The dataset of **user features** must contain identifiers for users, and use the same identifiers that were provided in the first column of the users-items-ratings dataset. The remaining columns can contain any number of

features that describe the users.

For an example, see the **Restaurant customer** dataset in Azure Machine Learning Studio (classic). A typical set of user features looks like this:

USERID	AMBIENCE	DRESS_PREFERENCE	TRANSPORT	SMOKER
U1004	family	informal	On foot	FALSE
U1005	friends	No preference	Car owner	TRUE

Item features dataset (optional)

The dataset of item features must contain item identifiers in its first column. The remaining columns can contain any number of descriptive features for the items.

For an example, see the Restaurant feature data dataset, provided in Azure Machine Learning Studio (classic) ([click Saved Datasets and then Samples](#)). A typical set of item features (in this case, the item is a restaurant) might look like this:

PLACEID	ALCOHOL	SMOKING_AREA	PRICE	RAMBIENCE
135106	Wine-Beer	none	low	family
132667	No_Alcohol_Served	permitted	medium	casual

Train the model

1. Add the **Train Matchbox Recommender** module to your experiment in Studio (classic), and connect it to the training data.
2. If you have a separate dataset of either user features and/or item features, connect them to the **Train Matchbox Recommender** module.
 - **User features dataset:** Connect the dataset that describes users to the second input.
 - **Item features dataset:** Connect the dataset that describes items to the third input.
3. For **Number of training batches**, type the number of batches for dividing the data during training.

Based on this value, the dataset of user-item-rating triples is divided into multiple parts or batches during training.

Because **Train Matchbox Recommender** runs batches in parallel, we recommend that the number of training batches be set to the number of available cores, if the entire training data fits into memory. Otherwise, the number of training batches should be set to the lowest multiple of the number of available cores for which the training data does fit into memory.

By default, the training data is split into four (4) batches. Only the dataset of user-item-rating triples is split. User or item features are not split because features do not need to be split.

4. For **Number of traits**, type the number of latent traits that should be learned for each user and item.

The higher the number of traits, the more accurate the predictions will typically be; however, training will be slower. The number of traits usually lies in the range 2 - 20.

5. For **Number of recommendation algorithm iterations**, indicate how many times the algorithm should process the input data.

The Matchbox recommender is trained using a message-passing algorithm that can iterate multiple times

over the input data. The higher this number, the more accurate the predictions; however, training is slower. Usually, the number of iterations is in the range 1 - 10.

6. Run the experiment, or select just the **Train Matchbox Recommender** module and select **Run selected**.

Examples

For examples of how recommendation models are used in Azure Machine Learning, see these sample experiments in the [Azure AI Gallery](#):

- [Movie recommender sample](#): Demonstrates how to train, evaluate, and score using a recommendation model.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Usage tips

- If a feature column has missing values, the mode of the non-missing values will be used as replacement for the missing values.
- All user and item features are re-scaled to have unit length, so that their maximum absolute value is 1. No translation is applied to the feature values, so as to maintain their sparsity.

Restrictions

Online update (or continuous training) of a recommendation model is not currently supported in Azure Machine Learning. If you want to capture user responses to recommendations and use those for improving the model, we suggest retraining the complete model periodically. Incremental training is not possible, but you can apply a sliding window to the training data to ensure that data volume is minimized while using the most recent data.

Estimating recommender memory usage

Currently, the lower bound of Matchbox's memory footprint is $16 * N \cdot (4 \cdot T + 2 \cdot R)$ bytes, where N refers to the number of user-item-rating triples in the training dataset, T to the number of latent traits, and R to the difference between the minimum and maximum rating (in the training dataset).

The size of a serialized Matchbox recommender model is approximately $16 * T \cdot (U \cdot R + I \cdot X + X \cdot Y)$ bytes, where U refers to the number of users, I to the number of items, X to the number of user features, and Y to the number of item features.

Expected inputs

NAME	TYPE	DESCRIPTION
Training dataset of user-item-rating triples	Data Table	Ratings of items by users, expressed as a triple (User, Item, Rating)
Training dataset of user features	Data Table	Dataset containing features that describe users (optional)
Training dataset of item features	Data Table	Dataset containing features that describe items (optional)

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Number of traits	≥ 0	Integer	10	Specify the number of traits to use with the recommender (optional)
Number of recommendation algorithm iterations	≥ 1	Integer	5	Specify the maximum number of iterations to perform while training the recommendation model (optional)
Number of training batches	≥ 1	Integer	4	Specify the number of training batches to use with the recommender (optional)

Outputs

NAME	TYPE	DESCRIPTION
Trained Matchbox recommender	ILearner interface	Trained Matchbox recommender

Exceptions

EXCEPTION	DESCRIPTION
Error 0022	Exception occurs if number of selected columns in input dataset does not equal to the expected number.
Error 0036	Exception occurs if multiple feature vectors were provided for a given user or item.
Error 0018	Exception occurs if input dataset is not valid.
Error 0035	Exception occurs if no features were provided for a given user or item.
Error 0034	Exception occurs if more than one rating exists for a given user-item pair.
Error 0053	Exception occurs in the case when there are no user features or items for Matchbox recommendations.
Error 0003	Exception occurs if one or more of inputs are null or empty.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Cross-Validate Model](#)

[Evaluate Recommender](#)

[Train](#)

[Score Matchbox Recommender](#)

Train Model

3/10/2021 • 6 minutes to read • [Edit Online](#)

Trains a classification or regression model in a supervised manner

Category: Machine Learning / Train

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Train Model** module in Azure Machine Learning Studio (classic) to train a classification or regression model. Training takes place after you have defined a model and set its parameters, and requires tagged data. You can also use **Train Model** to retrain an existing model with new data.

How the training process works

In Azure Machine Learning, creating and using a machine learning model is typically a three-step process.

1. You configure a model, by choosing a particular type of algorithm, and defining its parameters or hyperparameters. Choose any of the following model types:
 - [Classification models](#), based on neural networks, decision trees, and decision forests, and other algorithms.
 - [Regression models](#), which can include standard linear regression, or which use other algorithms, including neural networks and Bayesian regression.
2. Provide a dataset that is labeled, and has data compatible with the algorithm. Connect both the data and the model to **Train Model**.

What training produces is a specific binary format, the [iLearner](#), that encapsulates the statistical patterns learned from the data. You cannot directly modify or read this format; however, other modules in Studio (classic) can use this trained model.

You can also view properties of the model. For more information, see the [Results](#) section.

3. After training is completed, use the trained model with one of the [scoring modules](#), to make predictions on new data.

NOTE

Other specialized machine learning tasks require different training methods, and Studio (classic) provides separate training modules for them. For example, image detection, clustering, and anomaly detection all use custom training methods. **Train Model** is intended for use with regression and classification models only.

Supervised and unsupervised training

You might have heard the terms *supervised* or *unsupervised* learning. Training a classification or regression model with **Train Model** is a classic example of *supervised machine learning*. That means you must provide a dataset that contains historical data from which to learn patterns. The data should contain both the outcome (label) you are trying to predict, and related factors (variables). The machine learning model needs the outcomes to determine the features that best predict the outcomes.

During the training process, the data are sorted by outcomes and the algorithm extracts statistical patterns to build the model.

Unsupervised learning indicates either that the outcome is unknown, or you choose not to use known labels. For example, clustering algorithms usually employ unsupervised learning methods, but can use labels if available. Another example is topic modeling using [LDA](#). You cannot use **Train Model** with these algorithms.

TIP

New to machine learning? This tutorial walks you through the process of getting data, configuring an algorithm, training and then using a model: [Create your first machine learning experiment](#)

How to use Train Model

1. In Azure Machine Learning Studio (classic), configure a [classification model](#) or [regression model](#) models.

You can also train a custom model created by using [Create R Model](#).

2. Add the **Train Model** module to the experiment. You can find this module under the **Machine Learning** category. Expand **Train**, and then drag the **Train Model** module into your experiment.
3. On the left input, attach the untrained mode. Attach the training dataset to the right-hand input of **Train Model**.

The training dataset must contain a label column. Any rows without labels are ignored.

4. For **Label column**, click **Launch column selector**, and choose a single column that contains outcomes the model can use for training.
 - For classification problems, the label column must contain either **categorical** values or **discrete** values. Some examples might be a yes/no rating, a disease classification code or name, or an income group. If you pick a noncategorical column, the module will return an error during training.
 - For regression problems, the label column must contain **numeric** data that represents the response variable. Ideally the numeric data represents a continuous scale.

Examples might be a credit risk score, the projected time to failure for a hard drive, or the forecasted number of calls to a call center on a given day or time. If you do not choose a numeric column, you might get an error.

- If you do not specify which label column to use, Azure Machine Learning will try to infer which is the appropriate label column, by using the metadata of the dataset. If it picks the wrong column, use the column selector to correct it.

TIP

If you have trouble using the Column Selector, see the article [Select Columns in Dataset](#) for tips. It describes some common scenarios and tips for using the **WITH RULES** and **BY NAME** options.

5. Run the experiment. If you have a lot of data, this can take a while.

Results

After the model is trained:

- To view the model parameters and feature weights, right-click the output and select **Visualize**.
- To use the model in other experiments, right-click the model and select **Save Model**. Type a name for the model.

This saves the model as a snapshot that is not updated by repeated runs of the experiment.

- To use the model in predicting new values, connect it to the [Score Model](#) module, together with new input data.

Related tasks

If you need to train a type of model not supported by **Train Model**, there are several options:

- Create a custom scoring method using R script, or use one of the many R scoring packages available.
 - [Create R Model](#)
 - [Execute R Script](#)
- Write your own Python script to train and score a model, or use an existing Python library:
 - [Execute Python Script](#)
- Anomaly detection models
 - [Train Anomaly Detection Model](#) supports the anomaly detection modules in Studio (classic).
- Recommendation models
 - If your model uses the Matchbox recommend provided in Azure Machine Learning, use the [Train Matchbox Recommender](#) module.
 - If you are using a different algorithm for market basket analysis or recommendation, use its training methods, in [R script](#) or [Python script](#).
- Clustering models
 - Use [Train Clustering Model](#) for the included K-means algorithm.
 - For other clustering models, use [R script](#) or [Python script](#) modules to both configure and train the models.

Examples

For examples of how the **Train Model** module is used in machine learning experiments, see these experiments in the [Azure AI Gallery](#):

- [Retail Forecasting](#): Demonstrates how to build, train, and compare multiple models.
- [Flight Delay Prediction](#): Demonstrates how to train multiple related classification models.

Expected inputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	Untrained learner
Dataset	Data Table	Training data

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Label column	any	ColumnSelection		Select the column that contains the label or outcome column

Outputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	Trained learner

Exceptions

For a list of all module errors, see [Module Error Codes](#).

EXCEPTION	DESCRIPTION
Error 0032	Exception occurs if argument is not a number.
Error 0033	Exception occurs if argument is Infinity.
Error 0083	Exception occurs if dataset used for training cannot be used for concrete type of learner.
Error 0035	Exception occurs if no features were provided for a given user or item.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0020	Exception occurs if number of columns in some of the datasets passed to the module is too small.
Error 0021	Exception occurs if number of rows in some of the datasets passed to the module is too small.
Error 0013	Exception occurs if passed to module learner has invalid type.

See also

[Evaluate Model](#)

[A-Z Module List](#)

Tune Model Hyperparameters

3/10/2021 • 13 minutes to read • [Edit Online](#)

Performs a parameter sweep on a model to determine the optimum parameter settings

Category: [Machine Learning / Train](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Tune Model Hyperparameters](#) module in Azure Machine Learning Studio (classic), to determine the optimum hyperparameters for a given machine learning model. The module builds and tests multiple models, using different combinations of settings, and compares metrics over all models to get the combination of settings.

The terms *parameter* and *hyperparameter* can be confusing. The model's *parameters* are what you set in the properties pane. Basically, this module performs a *parameter sweep* over the specified parameter settings, and learns an optimal set of *hyperparameters*, which might be different for each specific decision tree, dataset, or regression method. The process of finding the optimal configuration is sometimes called *tuning*.

The module support two methods for finding the optimum settings for a model:

- **Integrated train and tune:** You configure a set of parameters to use, and then let the module iterate over multiple combinations, measuring accuracy until it finds a "best" model. With most learner modules, you can choose which parameters should be changed during the training process, and which should remain fixed.

Depending on how long you want the tuning process to run, you might decide to exhaustively test all combinations, or you could shorten the process by establishing a grid of parameter combinations and testing a randomized subset of the parameter grid.

- **Cross validation with tuning:** With this option, you divide your data into some number of folds and then build and test models on each fold. This method provides the best accuracy and can help find problems with the dataset; however, it takes longer to train.

Both methods generate a trained model that you can save for re-use.

Related tasks

- If you are building a clustering model, use [Sweep Clustering](#) to automatically determine the optimum number of clusters and other parameters.
- Before tuning, apply feature selection to determine the columns or variables that have the highest information value. For more information, see [Feature Selection](#).

How to configure Tune Model Hyperparameters

Generally, learning the optimal hyperparameters for a given machine learning model requires considerable experimentation. This module supports both the initial tuning process, and cross-validation to test model accuracy:

- [Find optimal model parameters using a parameter sweep](#)
- [Perform cross-validation during a parameter sweep](#)

Train a model using a parameter sweep

This section describes how to perform a basic parameter sweep, which trains a model by using the [Tune Model Hyperparameters](#) module.

1. Add the [Tune Model Hyperparameters](#) module to your experiment in Studio (classic).
2. Connect an untrained model (a model in the [iLearner](#) format) to the leftmost input.
3. Set the **Create trainer mode** option to **Parameter Range** and use the **Range Builder** to specify a range of values to use in the parameter sweep.

Almost all the [classification](#) and [regression](#) modules support an integrated parameter sweep. For those learners that do not support configuring a parameter range, only the available parameter values can be tested.

You can manually set the value for one or more parameters, and then sweep over the remaining parameters. This might save some time.

4. Add the dataset you want to use for training and connect it to the middle input of [Tune Model Hyperparameters](#).

Optionally, if you have a tagged dataset, you can connect it to the rightmost input port (**Optional validation dataset**). This lets you measure accuracy while training and tuning.

5. In the **Properties** pane of [Tune Model Hyperparameters](#), choose a value for **Parameter sweeping mode**. This option controls how the parameters are selected.

- **Entire grid:** When you select this option, the module loops over a grid predefined by the system, to try different combinations and identify the best learner. This option is useful for cases where you don't know what the best parameter settings might be and want to try all possible combination of values.

You can also reduce the size of the grid and run a **random grid** sweep. Research has shown that this method yields the same results, but is more efficient computationally.

- **Random sweep:** When you select this option, the module will randomly select parameter values over a system-defined range. You must specify the maximum number of runs that you want the module to execute. This option is useful for cases where you want to increase model performance using the metrics of your choice but still conserve computing resources.

6. For **Label column**, launch the column selector to choose a single label column.

7. Choose a single metric to use when **ranking** the models.

When you run a parameter sweep, all applicable metrics for the model type are calculated and are returned in the **Sweep results** report. Separate metrics are used for regression and classification models.

However, the metric you choose determines how the models are ranked. Only the top model, as ranked by the chosen metric, is output as a trained model to use for scoring.

8. For **Random seed**, type a number to use when initializing the parameter sweep.

If you are training a model that supports an integrated parameter sweep, you can also set a range of seed values to use and iterate over the random seeds as well. This can be useful for avoiding bias introduced by seed selection.

9. Run the experiment.

Results of hyperparameter tuning

When training is complete:

- To view a set of accuracy metrics for the best model, right-click the module, select **Sweep results**, and then select **Visualize**.

All accuracy metrics applicable to the model type are output, but the metric that you selected for ranking determines which model is considered "best". Metrics are generated only for the top-ranked model.

- To view the settings derived for the "best" model, right-click the module, select **Trained best model**, and then click **Visualize**. The report includes parameter settings and feature weights for the input columns.

- To use the model for scoring in other experiments, without having to repeat the tuning process, right-click the model output and select **Save as Trained Model**.

Perform cross-validation with a parameter sweep

This section describes how to combine a parameter sweep with cross-validation. This process takes longer, but you can specify the number of folds, and you get the maximum amount of information about your dataset and the possible models.

1. Add the [Partition and Sample](#) module to your experiment, and connect the training data.
2. Choose the **Assign to Folds** option and specify some number of folds to divide the data into. If you don't specify a number, by default 10 folds are used. Rows are apportioned randomly into these folds, without replacement.
3. To balance the sampling on some column, set the **Stratified split** to **TRUE**, and then select the *strata column*. For example, if you have an imbalanced dataset, you might want to divide the dataset such that each fold gets the same number of minority cases.
4. Add the [Tune Model Hyperparameters](#) module to your experiment.
5. Connect one of the machine learning modules in [this category](#) to the left-hand input of [Tune Model Hyperparameters](#).

6. In the **Properties** pane for the learner, set the **Create trainer mode** option to **Parameter Range** and use the **Range Builder** to specify a range of values to use in the parameter sweep.

You don't need to specify a range for all values. You can manually set the value for some parameters, and then sweep over the remaining parameters. This might save some time.

For a list of learners that don't support this option, see the [Technical Notes](#) section.

7. Connect the output of [Partition and Sample](#) to the labeled **Training dataset** input of [Tune Model Hyperparameters](#).
8. Optionally, you can connect a validation dataset to the rightmost input of [Tune Model Hyperparameters](#). For cross-validation, you need only a training dataset.
9. In the **Properties** pane of [Tune Model Hyperparameters](#), indicate whether you want to perform a random sweep or a grid sweep. A grid sweep is exhaustive, but more time-consuming. A random parameter search can get good results without taking quite so much time.

Maximum number of runs on random sweep: If you choose a random sweep, you can specify how many times the model should be trained, using a random combination of parameter values.

Maximum number of runs on random grid: This option also controls the number of iterations over a random sampling of parameter values, but the values are not generated randomly from the specified range; instead, a matrix is created of all possible combinations of parameter values and a random sampling is taken over the matrix. This method is more efficient and less prone to regional oversampling or undersampling.

TIP

For a more in-depth discussion of these options, see the [Technical notes](#) section.

10. Choose a single label column.
11. Choose a **single** metric to use in ranking the model. Many metrics are computed, so select the most important one to use in ordering the results.
12. For **Random seed**, type a number to use when initializing the parameter sweep.

If you are training a model that supports an integrated parameter sweep, you can also set a range of seed values to use and iterate over the random seeds as well. This is optional, but can be useful for avoiding bias introduced by seed selection.

13. Add the [Cross-Validate Model](#) module. Connect the output of [Partition and Sample](#) to the **Dataset** input,

and connect the output of [Tune Model Hyperparameters](#) to the **Untrained model** input.

14. Run the experiment.

Results of cross-validation

When cross-validation is complete:

- To view the evaluation results, right-click the module, select **Evaluation results by fold**, and then select **Visualize**.

The accuracy metrics are calculated from the cross-validation pass, and may vary slightly depending on how many folds you selected.

- To see how the dataset was divided, and how the "best" model would score each row in the dataset, right-click the module, select **Scored results**, and then select **Visualize**.
- If you save this dataset for later re-use, the fold assignments are preserved. For example, the saved dataset might look like this:

FOLD ASSIGNMENTS	CLASS	AGE(1ST FEATURE COLUMN)
2	0	35
1	1	17
3	0	62

- To get the parameter settings for the "best" model, right-click [Tune Model Hyperparameters](#)

Examples

For examples of how this module is used, see the [Azure AI Gallery](#):

- [Prediction of student performance](#): Uses the [Two-Class Boosted Decision Tree](#) algorithm with different parameters to generate a model with the best possible root mean squared error (RMSE).
- [Learning with Counts: Binary Classification](#): Generates a compact set of features using count-based learning, and then applies a parameter sweep to find the best model parameters.
- [Binary Classification: Network intrusion detection](#): Uses [Tune Model Hyperparameters](#) in cross-validation mode, with a custom split into five folds, to find the best hyperparameters for a [Two-Class Logistic Regression](#) model.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

How a parameter sweep works

This section describes how parameter sweep works in general, and how the options in this module interact.

When you set up a parameter sweep, you define the scope of your search, to use either a finite number of parameters selected randomly, or an exhaustive search over a parameter space you define.

- Random sweep**: This option trains a model using a set number of iterations.

You specify a range of values to iterate over, and the module uses a randomly chosen subset of those values. Values are chosen with replacement, meaning that numbers previously chosen at random are not removed from the pool of available numbers. Thus, the chance of any value being selected remains the same across all passes.

- Grid sweep**: This option creates a matrix, or grid, that includes every combination of the parameters in the value range you specify. When you start tuning with this module, multiple models are trained using combinations of these parameters.
- Entire grid**: The option to use the entire grid means just that: each and every combination is tested. This option can be considered the most thorough, but requires the most time.

- **Random grid:** If you select this option, the matrix of all combinations is calculated and values are sampled from the matrix, over the number of iterations you specified.

Recent research has shown that random sweeps can perform better than grid sweeps.

Controlling the length and complexity of training

Iterating over many combinations of settings can be time-consuming, so the module provides several ways to constrain the process:

- Limit the number of iterations used to test a model
- Limit the parameter space
- Limit both the number of iterations and the parameter space

We recommend that you experiment with the settings to determine the most efficient method of training on a particular dataset and model.

Choosing an evaluation metric

A report containing the accuracy for each model is presented at the end so that you can review the metric results. A uniform set of metrics is used for all classification models, and a different set of metrics is used for regression models. However, during training, you must choose a **single** metric to use in ranking the models that are generated during the tuning process. You might find that the best metric varies, depending on your business problem, and the cost of false positives and false negatives.

For more information, see [How to evaluate model performance in Azure Machine Learning](#)

Metrics used for classification

- **Accuracy** The proportion of true results to total cases.
- **Precision** The proportion of true results to positive results.
- **Recall** The fraction of all correct results over all results.
- **F-score** A measure that balances precision and recall.
- **AUC** A value that represents the area under the curve when false positives are plotted on the x-axis and true positives are plotted on the y-axis.
- **Average Log Loss** The difference between two probability distributions: the true one, and the one in the model.
- **Train Log Loss** The improvement provided by the model over a random prediction.

Metrics used for regression

- **Mean absolute error** Averages all the error in the model, where error means the distance of the predicted value from the true value. Often abbreviated as **MAE**.
- **Root of mean squared error** Measures the average of the squares of the errors, and then takes the root of that value. Often abbreviated as **RMSE**.
- **Relative absolute error** Represents the error as a percentage of the true value.
- **Relative squared error** Normalizes the total squared error by dividing by the total squared error of the predicted values.
- **Coefficient of determination** A single number that indicates how well data fits a model. A value of 1 means that the model exactly matches the data; a value of 0 means that the data is random or otherwise cannot be fit to the model. Often referred to as r^2 , R^2 , or **r-squared**.

Modules that do not support a parameter sweep

Almost all learners in Azure Machine Learning support cross-validation with an integrated parameter sweep, which lets you choose the parameters to experiment with. If the learner doesn't support setting a range of values, you can still use it in cross-validation. In this case, some range of allowed values is selected for the sweep.

The following learners do not support setting a range of values to use in a parameter sweep:

- [Two-Class Bayes Point Machine](#)

- Bayesian Linear Regression

Expected inputs

NAME	TYPE	DESCRIPTION
Untrained model	ILearner interface	Untrained model for parameter sweep
Training dataset	Data Table	Input dataset for training
Validation dataset	Data Table	Input dataset for validation (for Train/Test validation mode). This input is optional.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Specify parameter sweeping mode	List	Sweep Methods	Random sweep	Sweep entire grid on parameter space, or sweep with using a limited number of sample runs
Maximum number of runs on random sweep	[1;10000]	Integer	5	Execute maximum number of runs using random sweep
Random seed	any	Integer	0	Provide a value to seed the random number generator
Label column	any	ColumnSelection		Label column
Metric for measuring performance for classification	List	Binary Classification Metric Type	Accuracy	Select the metric used for evaluating classification models
Metric for measuring performance for regression	List	RegressionMetric Type	Mean absolute error	Select the metric used for evaluating regression models

Outputs

NAME	TYPE	DESCRIPTION
Sweep results	Data Table	Results metric for parameter sweep runs
Trained best model	ILearner interface	Model with best performance on the training dataset

See also

[A-Z Module List](#)
[Train](#)
[Cross-Validate Model](#)

OpenCV Library Modules

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that support use of the Open Source Computer Vision (OpenCV) Library.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

OpenCV is an open-source library that supports a variety of image processing and image recognition tasks. For more information, see [the OpenCV website](#).

The modules in Machine Learning Studio (classic) provide a way for you to easily incorporate the OpenCV Library into your machine learning experiments.

For additional image recognition features, see the image APIs published as part of [Microsoft Cognitive Services](#):

- [Face API](#). Detects faces and analyzes critical facial attributes, including emotion.
- [Computer Vision API](#). Supports domain detection, identification of adult content, image tagging, and image type or color analysis.
- [Bing Image Search](#). Gets images for machine learning projects by searching by type, color, region, and other attributes.

List of modules

The OpenCV Library category includes these modules:

- [Import Images](#): Loads images from Azure Blob storage into a dataset.
- [Pretrained Cascade Image Classification](#): Creates a pretrained image classification model for frontal faces by using the OpenCV Library.

See also

- [Regression](#)
- [Classification](#)
- [Clustering](#)
- [Text Analytics](#)
- [Module Categories and Descriptions](#)

Import Images

3/10/2021 • 4 minutes to read • [Edit Online](#)

Loads images from Azure BLOB Storage into a dataset

Category: [OpenCV Library Modules](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Import Images](#) module in Azure Machine Learning Studio (classic), to get multiple images from Azure Blob storage and create an image dataset from them.

When you use this module to load images from blob storage into your workspace, each image is converted to a series of numeric values for the red, green, and blue channels, together with the image file name. A dataset of such images consists of multiple rows in a table, each with a different set of RGB values and corresponding image file names. For instructions about how to prepare your images and connect to blob storage, see [How to Import Images](#).

After you have converted all your images, you can then pass this dataset to the [Score Model](#) module, and connect a pre-trained image classification model to predict the image type.

You can import any kind of images used for machine learning; however, there are limitations, including the types and size of images that can be processed, see the [Technical notes](#) section.

How to use Import Images

This example assumes that you have uploaded multiple images to your account in Azure blob storage. The images are in a container designated for that purpose only. As a rule, each image must be fairly small and have the same dimensions and color channels. For a detailed list of requirements that apply to images, see the [Technical notes](#) section.

1. Add the [Import Images](#) module to your experiment in Studio (classic).
2. Add the [Pretrained Cascade Image Classification](#) and the [Score Model](#) module.
3. In the [Import Images](#) module, configure the location of the images, and provide the authentication method, private or public:
 - If the image set is in a blob that has been configured for public access through [Shared Access Signatures](#)(SAS), type the URL to the container that holds the images.
 - If the images are stored in a private account in Azure storage, select **Account**, and then type the account name as it appears in the management portal. Then, paste in the primary or secondary account key.
 - For **Path to container**, type just the container name, and no other path elements.

4. Connect the output of [Import Images](#) to the [Score Model](#) module.

5. Run the experiment.

Results

Each row of the output dataset contains data from one image. The rows are sorted alphabetically by image name, and the columns contain the following information, in this order:

- The first column contains image names.
- All other columns contain flattened data from the red, green, and blue color channels, in that order.
- The transparency channel is ignored.

Depending on the color depth of the image and the image format, there could be many thousands of columns for a single image. Therefore, to view the results of the experiment, we recommend that you add the [Select Columns in Dataset](#) module, and select only these columns:

- Image Name
- Scored Labels
- Scored Probabilities

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions

Supported image formats

The [Import Images](#) module determines the type of an image by reading the first few bytes of the content, not by the file extension. Based on that information, it determines whether the image is one of the supported image formats.

- Windows bitmap files: .bmp, .dib
- JPEG files: jpeg, jpg, jpe
- JPEG 2000 files: jp2
- Portable Network Graphics: .png
- Portable image format: .pbm, .pgm, .ppm
- Sun Raster: .sr, .ras
- TIFF files: .tiff, .tif

Image requirements

The following requirements apply to images processed by the [Import Images](#) module:

- All images must be the same shape.
- All images must have the same color channels. For example, you cannot mix grayscale images with RGB images.
- There is a limit of 65536 pixels per image. However, the number of images is not limited.
- If you specify a blob container as the source, the container must not contain other types of data. Ensure that the container contains only images before running the module.

Other restrictions

- If you intend to use the [Pretrained Cascade Image Classification](#) module, be aware that it currently supports only recognition of faces in frontal view; other image classifiers are not yet available.
- You cannot use image datasets with these modules: [Train](#), [Cross-Validate Model](#).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Please specify authentication type	List	AuthenticationType	Account	Public or Shared Access Signature (SAS) URI or user credentials
URI	Any	String	none	Uniform Resource Identifier with SAS or public access
Account name	Any	String	none	Name of the Azure Storage account
Account key	Any	SecureString	none	Key associated with the Azure Storage account
Path to container, directory or blob	Any	String	none	Path to blob or name of table

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with downloaded images

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more inputs are null or empty.
Error 0029	Exception occurs when invalid URI is passed.
Error 0009	Exception occurs if the Azure storage account name or container name is specified incorrectly.
Error 0015	Exception occurs if the database connection has failed.
Error 0030	Exception occurs when it is not possible to download a file.
Error 0049	Exception occurs when it is not possible to parse a file.
Error 0048	Exception occurs when it is not possible to open a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Pretrained Cascade Image Classification](#)

[A-Z Module List](#)

Pretrained Cascade Image Classification

3/10/2021 • 6 minutes to read • [Edit Online](#)

Creates a pretrained image classification model for frontal faces using the OpenCV Library

Category: [OpenCV Library Modules](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Pretrained Cascade Image Classification** module in Azure Machine Learning Studio (classic), to detect faces in images.

The model is based on the [OpenCV](#) library. The OpenCV Library provides a list of predefined models, each optimized to detect a particular type of object.

More about the pre-trained model

This image recognition model has already been trained on a large corpus of images that is widely used for image recognition tasks. This particular classification model has been optimized for facial detection, and uses the Viola-Jones object detection algorithm. The purpose of the model is to identify images that contain a human face in frontal view.

Although currently only one OpenCV image classification model is provided, additional pretrained models might be available in later releases.

Using a pre-trained model

If you have a set of images that you would like to analyze, you provide them as input to the [Score Model](#) module as described in this topic, and attach this module, which provides the pretrained OpenCV library model.

The [Score Model](#) module uses the image classification model to determine whether the image contains a human face, and returns a probability score for each image used as input.

Models based on **Pretrained Cascade Image Classification** cannot be retrained on new image data.

The format in which the model is stored is incompatible with the [Train Model](#) and [Cross-Validate Model](#) modules.

How to configure Pretrained Cascade Image Classification

The image classification model in Azure Machine Learning has already been trained using a large dataset and is optimized for a specific image type. Therefore, all you need to do is provide a set of images as a *scoring dataset*. As an output, the module generates a score that indicates whether each image contains the target image type.

1. Prepare and import the dataset of images you plan to use in scoring. In general, all images in the dataset should be the same size.

You add the images to your experiment by using the [Import Images](#) module. Read the help for [Import](#)

[Images](#) closely to ensure that the images you use meet the requirements. You must also ensure that the images are accessible in the defined storage option.

2. Add the **Pretrained Cascade Image Classification** module to your experiment in Studio (classic). You can find this module in the **OpenCV Library** category.
3. Select one of the pre-trained classifiers from the list in **Pre-trained classifier**.

Currently, only one classifier is available: **Frontal face**, which is selected by default.

4. **Scale factor:** Type a value that specifies how much the image size is reduced at each image scale.

In the OpenCV Library, the classifier is designed so that it can be easily “resized” in order to be able to find the objects of interest at different sizes. This is more efficient than resizing the image itself. Thus, to find an object of an unknown size in the image the scan procedure should be done several times at different scales.

We recommend that you try different scaling factors to see which provides the best image classification results.

5. **Minimum number of neighbors:** Type a whole number that represents the minimum number of overlapping rectangles that are required to detect that a face is included in a region.

In the OpenCV library, the classifier detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles. The *neighbors* parameter controls how many possible matches are required to qualify as a detected face or feature. Thus, increasing this value tends to increase precision at the cost of coverage.

For examples of how neighbors are calculated, see this article in the OpenCv Library documentation: [Eigenfaces in OpenCV](#)

6. Optionally, you can use the following settings to specify image size to the model so that it can make better predictions. Images that do not fit the requirements are eliminated:

- **Minimum height:** Type the pixel height of the smallest image. If you specify a value for this property, images smaller than this are ignored.
- **Maximum height:** Type the pixel width of the largest image. If you specify a value for this property, images larger than this are ignored.
- **Minimum width:** Type the pixel width of the smallest image. If you specify a value for this property, images smaller than this are ignored.
- **Maximum width:** Type the pixel width of the largest image. If you specify a value for this property, images larger than this are ignored.

7. Connect the image dataset used for scoring.

8. Add the **Score Model** module to your experiment, and connect the pre-trained image classifier, and your dataset of images.

9. Run the experiment.

Results

The output of **Score Model** includes the image name, the scored label, and the probability score for the label (either 0 or 1). The classifier outputs a “1” if the image is likely to show the object (a face), and “0” otherwise. For example:

IMAGE NAME	SCORED LABELS	SCORED PROBABILITIES
MAN001.png	TRUE	1
TABLE001.PNG	FALSE	0
CHAIR001.PNG	FALSE	0

TIP

The output also contains the RGB values for all color channels in the dataset. Therefore, to view the data more easily, we recommend that in your experiment you use [Select Columns in Dataset](#) to output just the result columns.

Technical notes

The facial recognition model provided by this module is based on the Viola-Jones face detection algorithm. For more information, see these resources:

- This video explains the basic concepts of facial recognition, including a definition of *Haar features* and how they are used in facial detection: [Facial Detection - Part 1](#)
- This Wikipedia article describes the method used for the classifier, based on the paper by Navneet Dalal and Bill Triggs: [Histogram of oriented gradients](#)
- For the documentation of the face recognition algorithm provided in the OpenCV library, see [Cascade Classifier](#).

NOTE

This module does not output the full collection of information produced by the OpenCV library. In particular, this module only outputs the prediction of whether a face is present or not, and does not include the coordinates of the face or any other information.

If you need this additional information, consider using other libraries, such as the [Face API](#) provided by [Microsoft Cognitive Services](#).

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Pre-trained classifier	List	PretrainedClassifier	Frontal face	Pretrained classifier from standard OpenCV distribution.
Scale factor	>=1.0000000000000002	Float	1.1	Parameter that specifies how much the image size is reduced at each image scale.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Minimum number of neighbors	$>=0$	Integer	3	Parameter that specifies how many neighbors each candidate rectangle should have to retain it.
Minimum height	$>=1$	Integer	100	Minimum possible object height (in pixels). Objects smaller than this are ignored. The parameter is optional.
Minimum width	$>=1$	Integer	100	Minimum possible object width (in pixels). Objects smaller than this are ignored. The parameter is optional.
Maximum height	$>=1$	Integer	200	Maximum possible object height (in pixels). Objects larger than this are ignored. The parameter is optional.
Maximum width	$>=1$	Integer	200	Maximum possible object width (in pixels). Objects larger than this are ignored. The parameter is optional.

Outputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	Trained binary classification model

Exceptions

EXCEPTION	DESCRIPTION
Error 0005	Exception occurs if parameter is less than a specific value.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Import Images](#)

[Pretrained Cascade Image Classification](#)

Python Language Modules

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the modules provided in Azure Machine Learning Studio (classic) that support running custom Python code in a machine learning experiment.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The Python environment in Azure Machine Learning uses the Anaconda environment, which is easy to use and also includes some of the more important and popular Python packages, including `NumPy`, `SciPy`, and `scikit-learn`. To run Python code using these packages, just write your code in the text editor of the [Execute Python Script](#) module.

For more information about the Python environment, see [Microsoft Azure Machine Learning Python Client Library](#).

For more information about how you can use Python code in Azure Machine Learning, see these resources:

- [Execute Python Scripts in Azure Machine Learning](#)
- [Access datasets with Python using the Azure Machine Learning client library](#)

List of modules

The Python Language Modules category includes the following module:

- [Execute Python Script](#): Executes a Python script from an Azure Machine Learning experiment

See also

[R Language Modules](#)

[Module Categories and Descriptions](#)

Execute Python Script

3/10/2021 • 7 minutes to read • [Edit Online](#)

Executes a Python script from an Azure Machine Learning experiment

Category: [Python Language Modules](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Execute Python Script** module in Azure Machine Learning Studio (classic) to run Python code. For more information about the architecture and design principles of Python in Studio (classic), see [the following article](#).

With Python, you can perform tasks that aren't currently supported by existing Studio (classic) modules such as:

- Visualizing data using `matplotlib`
- Using Python libraries to enumerate datasets and models in your workspace
- Reading, loading, and manipulating data from sources not supported by the [Import Data](#) module

Azure Machine Learning Studio (classic) uses the Anaconda distribution of Python, which includes many common utilities for data processing.

How to use Execute Python Script

The **Execute Python Script** module contains sample Python code that you can use as a starting point. To configure the **Execute Python Script** module, you provide a set of inputs and Python code to execute in the **Python script** text box.

1. Add the **Execute Python Script** module to your experiment.
2. Scroll to the bottom of the **Properties** pane, and for **Python Version**, select the version of the Python libraries and runtime to use in the script.
 - Anaconda 2.0 distribution for Python 2.7.7
 - Anaconda 4.0 distribution for Python 2.7.11
 - Anaconda 4.0 distribution for Python 3.5 (default)

We recommend that you set the version before typing any new code. If you change the version later, a prompt asks you to acknowledge the change.

IMPORTANT

If you use multiple instances of the **Execute Python Script** module in your experiment, you must choose a single version of Python for all modules in the experiment.

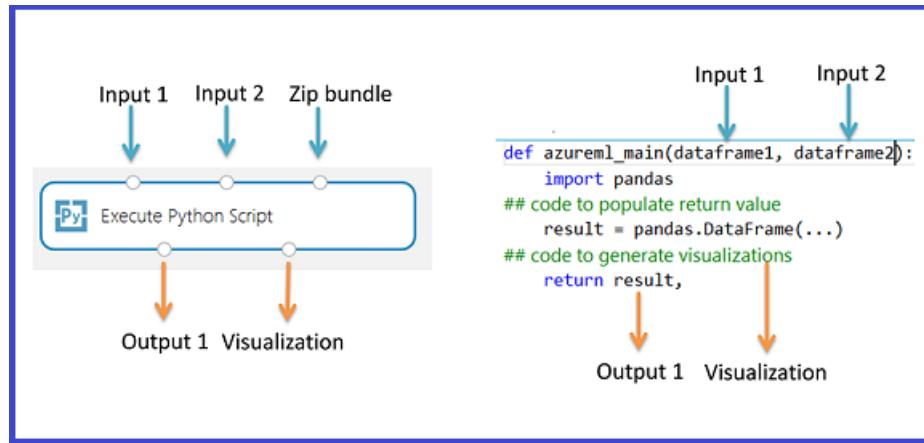
3. Add and connect on **Dataset1** any datasets from Studio (classic) that you want to use for input.

Reference this dataset in your Python script as **DataFrame1**.

Use of a dataset is optional, if you want to generate data using Python, or use Python code to import the data directly into the module.

This module supports addition of a second Studio (classic) dataset on **Dataset2**. Reference the second dataset in your Python script as **DataFrame2**.

Datasets stored in Studio (classic) are automatically converted to **pandas** data.frames when loaded with this module.



4. To include new Python packages or code, add the zipped file containing these custom resources on **Script bundle**. The input to **Script bundle** must be a zipped file already uploaded to your workspace. For more information about how to prepare and upload these resources, see [Unpack Zipped Data](#).

Any file contained in the uploaded zipped archive can be used during experiment execution. If the archive includes a directory structure, the structure is preserved, but you must prepend a directory called **src** to the path.

5. In the **Python script** text box, type or paste valid Python script.

The **Python script** text box is pre-populated with some instructions in comments, and sample code for data access and output. You must edit or replace this code. Be sure to follow Python conventions about indentation and casing.

- The script must contain a function named `azureml_main` as the entry point for this module.
- The entry point function can contain up to two input arguments: `Param<dataframe1>` and `Param<dataframe2>`.
- Zipped files connected to the third input port are unzipped and stored in the directory, `.\Script Bundle`, which is also added to the Python `sys.path`.

Therefore, if your zip file contains `mymodule.py`, import it using `import mymodule`.

- A single dataset can be returned to Studio (classic), which must be a sequence of type `pandas.DataFrame`. You can create other outputs in your Python code and write them directly to Azure storage, or create visualizations using the **Python device**.

6. Run the experiment, or select the module and click **Run selected** to run just the Python script.

All of the data and code is loaded into a virtual machine, and run using the specified Python environment.

Results

The module returns these outputs:

- **Results Dataset**. The results of any computations performed by the embedded Python code must be

provided as a pandas data.frame, which is automatically converted to the Azure Machine Learning dataset format, so that you can use the results with other modules in the experiment. The module is limited to a single dataset as output. For more information, see [Data Table](#).

- **Python Device.** This output supports both console output and display of PNG graphics using the Python interpreter.

How to attach script resources

The **Execute Python Script** module supports arbitrary Python script files as inputs, provided they are prepared in advance and uploaded to your workspace as part of a .ZIP file.

Upload a ZIP file containing Python code to your workspace

1. In the experiment area of Azure Machine Learning Studio (classic), click **Datasets**, and then click **New**.
2. Select the option, **From local file**.
3. In the **Upload a new dataset** dialog box, click the dropdown list for **Select a type for the new dataset**, and select the **Zip file (.zip)** option.
4. Click **Browse** to locate the zipped file.
5. Type a new name for use in the workspace. The name you assign to the dataset becomes the name of the folder in your workspace where the contained files are extracted.
6. After you have uploaded the zipped package to Studio (classic), verify that the zipped file is available in the **Saved Datasets** list, and then connect the dataset to the **Script Bundle** input port.

All files that are contained in the ZIP file are available for use during run time: for example, sample data, scripts, or new Python packages.

If your zipped file contains any libraries that are not already installed in Azure Machine Learning Studio (classic), you must install the Python library package as part of your custom script.

If there was a directory structure present, it is preserved. However, you must alter your code to prepend the directory **src** to the path.

Debugging Python code

The **Execute Python Script** module works best when the code has been factored as a function with clearly defined inputs and outputs, rather than a sequence of loosely related executable statements.

This Python module does not support features such as Intellisense and debugging. If the module fails at runtime, you can view some error details in the output log for the module. However, the full Python stack trace is not available. Thus we recommend that users develop and debug their Python scripts in a different environment and then import the code into the module.

Some common problems that you can look for:

- Check the data types in the data frame you are returning back from `azureml_main`. Errors are likely if columns contain data types other than numeric types and strings.
- Remove NA values from your dataset, using `dataframe.dropna()` on export from Python script. When preparing your data, use the [Clean Missing Data](#) module.
- Check your embedded code for indentation and whitespace errors. If you get the error, "IndentationError: expected an indented block", see these resources for guidance:
 - [Python Reference - Indentation](#)

- [Style Guide for Python Code](#)

Known limitations

- The Python runtime is sandboxed and does not allow access to the network or to the local file system in a persistent manner.
- All files saved locally are isolated and deleted once the module finishes. The Python code cannot access most directories on the machine it runs on, the exception being the current directory and its subdirectories.

When you provide a zipped file as resource, the files are copied from your workspace to the experiment execution space, unpacked, and then used. Copying and unpacking resources can consume memory.

- The module can output a single data frame. It's not possible to return arbitrary Python objects such as trained models directly back to the Studio (classic) runtime. However, you can write objects to storage or to the workspace. Another option is to use `pickle` to serialize multiple objects into a byte array and then return the array inside a data frame.

Examples

For examples of integrating Python script with Studio (classic) experiments, see these resources in the [Azure AI Gallery](#):

- [Execute Python Script](#): Use text tokenization, stemming, and other natural language processing using the **Execute Python Script** module.
- [Custom R and Python scripts in Azure ML](#): Walks you through the process of adding custom code a(either R or Python), processing data, and visualizing the results.
- [Analyzing PyPI Data to Determine Python 3 Support](#): Estimate the point when demand for Python 3 outstrips that for Python 2.7 using python.

See also

[R Language Modules](#)

R Language Modules

3/10/2021 • 6 minutes to read • [Edit Online](#)

This article lists the modules in Azure Machine Learning Studio (classic) that support running R code. These modules make it easier to publish R models in production, and to use the experience of the R language community to solve real-world problems.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

This article also describes some general requirements for using R in Machine Language Studio (classic), and lists known issues and tips.

List of modules

The **R Language Modules** category includes the following modules:

- [Execute R Script](#): Runs an R script from a Machine Learning experiment.
- [Create R Model](#): Creates an R model by using custom resources.

Requirements when using R

Before using R script in Machine Learning Studio (classic), observe the following requirements:

- If you imported data that uses CSV or other formats, you cannot read the data directly in CSV format from your R code. Instead, use [Convert to Dataset](#) to prepare the data, before using it as input to an R module.
- When you attach any Machine Learning dataset as input to an R module, the dataset is automatically loaded into the R workspace as a data frame, with the variable name **dataset**.

However, you can define additional data frames, or change the name of the default dataset variable within your R script.

- The R modules run in a protected and isolated environment within your private workspace. Within your workspace, you can create data frames and variables for use by multiple modules.

However, you cannot load R data frames from a different workspace, or read variables created in a different workspace, even if that workspace is open in an Azure session. Also, you cannot use modules that have a Java dependency, or that require direct network access.

Optimization for R scoring tasks

The implementation of R in the Machine Learning Studio (classic) and workspace environment includes two principal components. One component coordinates script execution, and the other provides high-speed data access and scoring. The scoring component is optimized to enhance scalability and performance.

Therefore, R workspaces in Machine Learning Studio (classic) also support two kinds of scoring tasks, each

optimized for different requirements. You typically use scoring on a file-by-file basis when you are building an experiment. You typically use the request response service (RRS) for very fast scoring, when you are scoring as part of a web service.

R package and version support

Machine Learning Studio (classic) includes over 500 of the most popular R packages. The R packages that you can select from depend on which R version you select for your experiment:

- CRAN R
- Microsoft R Open (MRO 3.2.2 or MRO 3.4.4)

Whenever you create an experiment, you must choose a single R version to run on, for all modules in your experiment.

List of packages per version

For a list of the packages that are currently supported in Machine Learning, see [R Packages Supported by Azure Machine Learning](#).

You can also add the following code to an [Execute R Script](#) module in your experiment, and run it to get a dataset containing package names and versions. Be sure to set the R version in the module properties to generate the correct list for your intended environment.

```
data.set <- data.frame(installed.packages())
maml.mapOutputPort("data.set")
```

IMPORTANT

The packages that are supported in Machine Language Studio (classic) change frequently. If you have any doubts about whether an R package is supported, use the R code sample provided to get the complete list of packages in the current environment.

Extend experiments by using the R language

There are many ways that you can extend your experiment by using custom R script or by adding R packages. Here are some ideas to get you started:

- Use R code to perform custom math operations. For example, there are R packages to solve differential equations, generate random numbers, or run Monte Carlo simulations.
- Apply custom transformations for data. For example, you might use an R package to perform interpolation on time series data, or perform linguistic analysis.
- Work with different data sources. The R script modules support an additional set of inputs, which can include data files, in zipped format. You might use zipped data files, along with R packages designed for such data sources, to flatten hierarchical data into a flat data table. You might also use these to read data from Excel and other file formats.
- Use custom metrics for evaluation. For example, rather than use the functions provided in [Evaluate](#), you could import an R package, and then apply its metrics.

The following example demonstrates the overall process for how you can install new packages and use custom R code in your experiment.

Split columns by using R

Sometimes the data requires extensive manipulation to extract features. Suppose you have a text file that contains an ID followed by values and notes, all separated by spaces. Or suppose that your text file contains characters that are not supported by Machine Language Studio (classic).

There are several R packages that provide specialized functions for such tasks. The [splitstackshape library](#) package contains several useful functions for splitting multiple columns, even if each column has a different delimiter.

The following sample illustrates how to install the needed packages, and split apart columns. You would add this code to the **Execute R Script** module.

```
#install dependent packages
install.packages("src/concat.split.multiple/data.table_1.9.2.zip", lib=(".", repos = NULL, verbose = TRUE)
(success.data.table <- library("data.table", lib.loc = ".", logical.return = TRUE, verbose = TRUE))

install.packages("src/concat.split.multiple/plyr_1.8.1.zip", lib=(".", repos = NULL, verbose = TRUE)
(success.plyr <- library("plyr", lib.loc = ".", logical.return = TRUE, verbose = TRUE))

install.packages("src/concat.split.multiple/Rcpp_0.11.2.zip", lib=(".", repos = NULL, verbose = TRUE)
(success.Rcpp <- library("Rcpp", lib.loc = ".", logical.return = TRUE, verbose = TRUE))

install.packages("src/concat.split.multiple/reshape2_1.4.zip", lib=(".", repos = NULL, verbose = TRUE)
(success.reshape2 <- library("reshape2", lib.loc = ".", logical.return = TRUE, verbose = TRUE))

#install actual packages
install.packages("src/concat.split.multiple/splitstackshape_1.2.0.zip", lib=(".", repos = NULL, verbose = TRUE)
(success.splitstackshape <- library("splitstackshape", lib.loc = ".", logical.return = TRUE, verbose = TRUE))

#Load installed library
library(splitstackshape)

#Use library method to split & concat
data <- concat.split.multiple(maml.mapInputPort(1), c("TermsAcceptedUserClientIPAddress", "EmailAddress"),
c(".", "@"))

#print column names to console
colnames(data)

#Redirect data to output port
maml.mapOutputPort("data")
```

Additional resources

Begin with this tutorial that describes how to build a custom R module:

- [Extend Your Experiment with R](#)

This article discusses the differences between the two scoring engines in detail, and explains how you can choose a scoring method when you deploy your experiment as a web service:

- [Machine Learning: Consume Web Services](#)

This experiment in the Azure AI Gallery demonstrates how you can create a custom R module that does training, scoring, and evaluation:

- [Create R Model with Evaluation](#)

This article, published on R-Bloggers, demonstrates how you can create your own evaluation method in Machine Learning:

- [How to evaluate R models in Azure Machine Learning Studio \(classic\)](#)

More help with R

This site provides a categorized list of packages that you can search by keywords:

- [R documentation](#)

For additional R code samples and help with R and its applications, see these resources:

- [R Project](#): The official site for the R language.
- [Rseek](#): A search engine for R resources.
- [R-bloggers](#): An aggregation of blogs in the R community.
- [CRAN](#): The largest repository of R packages.
- [Quick-R](#): A good R tutorial.
- [Webinar: Learn How to Get Faster End Results from Your R Models](#)
- [Bioconductor](#): A large repository of R packages in bioinformatics.

See also

- [Python Language Modules](#)
- [Module Categories and Descriptions](#)

Execute R Script

3/10/2021 • 20 minutes to read • [Edit Online](#)

Executes an R script from an Azure Machine Learning Studio (classic) experiment

Category: [R Language Modules](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Execute R Script** module in Azure Machine Learning Studio (classic), to call and run R code in your experiments.

By adding R code to this module, you can perform a variety of customized tasks that are not available in Studio (classic). For example:

- Create custom data transformations
- Use your own metrics for evaluating predictions
- Build models using algorithms that are not implemented as standalone modules in Studio (classic)

R versions supported in Studio (classic)

Studio (classic) supports both the typical distribution of R that is available from CRAN, and Microsoft R Open (MRO), which includes all the base R packages, plus the [Revo packages](#).

You can specify which version of R to use in an experiment. However, you cannot install any other version of R into your workspace.

We recommend that you determine which packages you need before choosing a distribution of R. Some packages are not compatible with both CRAN R and Microsoft R Open.

NOTE

Currently, the [Create R Model](#) module is limited to specific version of R. Therefore, if you use a custom R model in your experiment, any **Execute R Script** modules in the same experiment must also use the same R version. Find the supported R version in the following article, [R Packages Supported by Azure Machine Learning Studio \(classic\)](#).

Supported R packages

The R environment in Azure Machine Learning has over 500 R packages already installed. Of course, not all are loaded by default, but you can easily load them as part of your R code.

To get a list of all current packages, add the following code to an **Execute R Script** module and run the module.

```
data.set <- data.frame(installed.packages())
maml.mapOutputPort("data.set")
```

This topic lists the packages that are supported in Azure Machine Learning, and their compatibility with CRAN R and Microsoft R Open, see [R Packages Supported by Azure Machine Learning Studio \(classic\)](#).

Installing new R packages

You install new R packages into your workspace by using the **Execute R Script** module. The packages must be uploaded in zipped format. When your experiment is loaded into an Azure runtime environment, the packages are unpacked and are added into the R environment in your experiment workspace. For more information, see [How to install new R packages](#)

Packages that have been unpacked are **not** persisted in the workspace when the experiment is not running. For this reason, any additional R packages that you plan to use must be available in your workspace, or in Azure storage, in zipped format.

Packages cannot be shared across separate instances of the **Execute R Script** module, because each module might be loaded into a different container at run time. However, you can share R objects between modules by exposing them as datasets. For more information, see [Pass R objects between modules](#).

Sample experiments

There are many examples of custom R script in the [Azure AI Gallery](#):

- [Student performance](#): Uses custom R script to combine the results of evaluations for multiple models into a single dataset. This sample also uses R code in the **Execute R Script** module to compute 16 time-dependent columns.
- [Breast cancer](#): Uses custom code in the **Execute R Script** module to replicate positive examples and to combine metrics.
- [Time series forecasting](#): This sample uses **Execute R Script** to generate custom metrics, and then combines them into a single table by using the [Add Rows](#) module.

How to configure Execute R Script

To configure the **Execute R Script** module, you provide a set of optional inputs and the R code that is to be run in the workspace.

You can also add files containing additional R code, if you prepare them in a zipped archive file for attachment to the **Script bundle** input.

To install any additional packages, include them in the zipped archive file.

1. Add the **Execute R Script** module to your experiment. You can find this module in Azure Machine Learning Studio (classic), in the **R Language Modules** group.
2. Connect any inputs needed by the script. Inputs can include data, R packages that you added to your workspace in zipped file format, and additional R code.
 - **Dataset1**: The first input is where you attach your main dataset (optional). The input dataset must be formatted as a CSV, TSV, or ARFF file, or you can connect an Azure Machine Learning dataset.
 - **Dataset2**: The second input (optional) supports addition of a second dataset. This dataset also must be formatted as a CSV, TSV, or ARFF file, or you can connect an Azure Machine Learning dataset.
 - **Script Bundle**: The third input, which is optional, takes a file in the .ZIP format. The zipped file can contain multiple files and multiple file types. For example, the zipped archive might contain R code in a script file, R objects for use by the script, an R package that itself was included in .ZIP format, or datasets in one of the supported formats.

3. Type R script into the **R Script** text box. This is the easiest way to work with the datasets on the input nodes.

To help you get started, the **R Script** text box is prepopulated with the following sample code, which you can edit or replace.

```
# Map 1-based optional input ports to variables
dataset1 <- maml.mapInputPort(1) # class: data.frame
dataset2 <- maml.mapInputPort(2) # class: data.frame

# Contents of optional Zip port are in ./src/
# source("src/yourfile.R");
# load("src/yourData.rdata");

# Sample operation
colnames(dataset2) <- c(dataset1['nombre_columna'])$nombre_columna;
data.set = dataset2;

# You'll see this output in the R Device port.
# It'll have your stdout, stderr and PNG graphics device(s).

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("data.set");
```

For more information about how to use inputs and write to outputs, see [R code samples](#) in this topic.

NOTE

R code that runs fine in external tools might need small changes to run in an Azure ML experiment. For example, input data that you provide in CSV format should be explicitly converted to a dataset before you can use it in your code. Data and column types used in the R language also differ in some ways from the data and column types used in Azure Machine Learning. For details, see the [Technical Notes](#) section.

Execute R Script module is running in a sandbox of R environment, it's **not** recommended to setup HTTP/SQL connections in this module.

4. **Random Seed:** Type a value to use inside the R environment as the random seed value. This parameter is equivalent to calling `set.seed(value)` in R code.

5. **R Version:** Select the version of R to load in the workspace.

- **CRAN R 3.1.0:** The Comprehensive R Archive Network Web site is the repository for the open source R language. For more information, see the [CRAN Web site](#).
- **Microsoft R Open 3.2.2:** MRO is the enhanced distribution of R from Microsoft Corporation. It is an open source platform based on the open source R engine and fully compatible with all R packages, scripts and applications that work with the same version of R. However, MRO provides improved performance in comparison to the standard R distribution due to its use of high-performance, multi-threaded math libraries. For more information, see [Microsoft R Open](#).
 - You cannot install any other version of R into your workspace.
 - Azure Machine Learning supports multiple versions of R, but only one version can be used in any experiment.

6. Run the experiment, or select the **Execute R Script** module and click **Run selected**.

Results

The module can return multiple outputs.

- To get a dataset back, your R code should return a single R data.frame.
- You can display images in the R graphics device, which is displayed in the Azure Machine Learning Studio (classic) log area.
- To persist images, you can write them to a file, or serialize them to a tabular format.
- You can save objects to your workspace.
- Standard messages and errors from R are returned to the module's log.

(1) Result Dataset

This output contains the data frame that is generated by the R code in the module.

You can output only one data frame. Other tabular objects must be converted to a data frame using R functions. The data frame output by the module's R code is automatically converted to the internal [Data Table](#) format.

- To verify that the returned object is compatible with Studio (classic), use `is.data.frame`, which must return True.
- To return other R objects, try serializing the object into a byte array, or use a function that returns the desired data as a `data.frame`.

(2) R Device

The R device supports both console output (standard output and standard error) and display of PNG graphics using the R interpreter.

- To view messages sent to the R console (Standard Output and Standard Error), right-click the module after it has finished running, select **R Device**, and select **Visualize**.
- To view graphics generated on the R Device port, right-click the module after it has finished running, select **R Device**, and select **Visualize**.

For example, the following image is generated by just a few lines of R code.

◀ Graphics



You can find this and related samples in the [Azure AI Gallery](#).

- To save images generated by the **Execute R Script** module, right-click the image and save a local copy. Or, you can use a call to one of the R graphics device functions to write the image file to the Azure blob storage account associated with the experiment, as described in [this example](#).

Sample R scripts and R tips

There are many ways that you can extend your experiment by using custom R script. This section provides sample code for some common tasks.

To get started using R in the **Execute R Script** module, see this video: [Using R in Azure Machine Learning Studio \(classic\)](#)

Add an R script as input

The **Execute R Script** module supports the use of arbitrary R script files as inputs, provided they are prepared in advance and uploaded to your workspace as part of the ZIP file.

1. To upload a ZIP file containing R code to your workspace, click **New**, click **Dataset**, and then select **From local file** and the **Zip file** option.
2. After you have uploaded the zipped package to Studio (classic), verify that the zipped file is available in the **Saved Datasets** list, and then connect the dataset to the **Script Bundle** input port.
3. If your zipped file contains any R package that is not already installed in Azure Machine Learning Studio (classic), you must install the R package as part of the custom code in the **Execute R Script** module. All files that are contained in the ZIP file are available during experiment run time.

If the script bundle file contained a directory structure, the structure is preserved. However, you must alter your code to prepend the directory **src** to the path.

Generate images, models, and other objects

If you need to generate an image, or any other arbitrary R object, you can serialize it into a byte array and then as a `data.frame` as shown in this example:

```
as.data.frame(as.integer(serializerserialize(g,con=NULL)));
```

Graph data frames from the <https://igraph.org/r/> library do not support serialization as a data frame. Instead, use the `get.data.frame` function in the `igraph` package to put the edge and vertex information into a data frame.

```
vertices <- get.data.frame(g, what="vertices")
```

You could then return the graph object as a `data.frame` that you can get from the **Execute R Script** module.

```
edges <- get.data.frame(g, what="edges")
```

Read from input and write to output

The following example demonstrates how to use input and output ports. It reads the input data as a table, and appends a copy of the table to itself, effectively doubling the size of the table. The result is then sent to the output port.

```
# Map existing dataset to first input port
dataset1 <- maml.mapInputPort(1) # class: data.frame
# Concatenate dataset1 to dataset 1
newdataset = rbind(dataset1, dataset1)
# Send the combined dataset to the output port
maml.mapOutputPort("newdataset");
```

Read a ZIP file as input

This example demonstrates how to add a dataset to Azure Machine Learning Studio (classic) in zipped format and then use the data as an input to the **Execute R Script** module.

1. Create the data file in CSV format, and name it "mydatafile.csv".
2. Create a .ZIP file and add the CSV file to the archive.
3. Upload the zipped file to your Azure Machine Learning workspace as described here: [Unpack Zipped Datasets](#).

4. Connect the resulting dataset to the **ScriptBundle** input of your **Execute R Script** module. In other words, do not unpack it yet!
5. Using the following code, read the CSV data from the zipped file. Specify the encoding that is used in the data file if necessary, to avoid errors later.

```
mydataset=read.csv("src/newdata.csv",encoding="UTF-8");
nrow(mydataset);
ncol(mydataset);
# Map new dataset to the first output port
maml.mapOutputPort("mydataset");
```

NOTE

All data passed to the **Execute R Script** module is converted to the `data.frame` format for use with your R code. This applies to any data that is compatible with the `DataTable` format used by Azure Machine Learning, including CSV files, ARFF files, and so on.

Replicate rows

This sample shows how to replicate the positive samples in a dataset by a factor of 20, to balance the sample.

```
dataset <- maml.mapInputPort(1)
data.set <- dataset[dataset[,1]==-1,]
pos <- dataset[dataset[,1]==1,]
for (i in 1:20) data.set <- rbind(data.set,pos)
row.names(data.set) <- NULL
maml.mapOutputPort("data.set")
```

Call a custom learner based on the Arules package

You can install new R packages to your Azure Machine Learning workspace by uploading them as a .ZIP file, as described [here](#). The following code demonstrates how to use the uploaded package.

1. Assume that the `arules` and `arulesViz` packages have already been added to the workspace.
2. Connect the uploaded .ZIP file to the third input port of the **Execute R Script** module.
3. In the **R Script** text box, use the following to call the *a priori* association rules algorithm provided by the R language package `Arules`, and apply the learner in a market basket analysis task.

```
library("arules")
library("arulesViz")
dataset <- read.transactions(file="src/SalesReport.csv", rm.duplicates= TRUE,
format="single",sep=",",cols =c(1,2))
#dataset <- sapply(dataset,as.factor)
basket <- apriori(dataset,parameter = list(sup = 0.5, conf = 0.9,target="rules"));
inspect(basket)
# if this is not NULL i.e. if there are rules
plot(basket)
```

Call a custom Naïve Bayes learner

This example shows how to call an R library that is not included in Studio (classic).

1. Upload a zipped file containing the `e1071` library to your workspace.
2. Connect the uploaded .ZIP file to the third input port of the **Execute R Script** module.
3. In the **R Script** text box, use the following code to implement the Naïve Bayes learner.

```
library(e1071)
features <- get.feature.columns(dataset)
labels   <- get.label.column(dataset)
train.data <- data.frame(features, labels)
feature.names <- get.feature.column.names(dataset)
names(train.data) <- c(feature.names, "Class")
model <- naiveBayes(Class ~ ., train.data)
```

Call a custom Naïve Bayes scorer

If you have an existing model created by the `e1071` library, you can call a custom scorer provided by the `e1071` library.

However, to perform scoring in a separate instance of the **Execute R Script** module, you must provide the zipped file containing the `e1071` library as an input to the scoring module as well, and load the library. That is because each module runs independently in a container.

```
library(e1071)
features <- get.feature.columns(dataset)
scores <- predict(model, features)
```

All R modules that are included inside a single experiment must use the same version of the R runtime. You cannot mix versions of R, such as using CRANR in one module and Microsoft R Open in another.

Write a graphics file

Although Studio (classic) supports the display of PNG files using the **R Device** output port, you might want to generate the results as a PDF file in a blob in Azure Storage to use for reporting.

This example demonstrates how to use the **Execute R Script** to generate a chart as a PDF file.

1. Add the **Execute R Script** to the experiment.
2. Create the basic PDF file as part of your R script, and return the Base64-encoded string of the PDF file from the **Execute R Script** module.

```
d <- maml.mapInputPort(1)
d$dteday <- as.numeric(d$dteday)
pdf()
plot(d)
dev.off()
library(caTools)
b64ePDF <- function(filename) {
    maxFileSizeInBytes <- 5 * 1024 * 1024 # 5 MB
    return(base64encode(readBin(filename, "raw", n = maxFileSizeInBytes)))
}
d2 <- data.frame(pdf = b64ePDF("Rpplots.pdf"))

maml.mapOutputPort("d2");
```

3. Pass this output to a **Export Data** module, and save the binary values to Azure blob storage.

Pass R objects between Execute R Script modules

You can pass R objects between instances of the **Execute R Script** module by using the internal serialization mechanism. This example assumes that you want to move the R object named `A` between two **Execute R Script** modules.

1. Add the first **Execute R Script** module to your experiment, and type the following code in the **R Script**

text box to create a serialized object **A** as a column in the module's output Data Table:

```
serialized <- as.integer(serialise(A,NULL))
data.set <- data.frame(serialized,stringsAsFactors=FALSE)
maml.mapOutputPort("data.set")
```

The explicit conversion to integer type is required because the serialization function outputs data in the R **Raw** format, which is not supported by Azure Machine Learning.

2. Add a second instance of the **Execute R Script** module, and connect it to the output port of the previous module.
3. Type the following code in the **R Script** text box to extract object **A** from the input Data Table.

```
dataset <- maml.mapInputPort(1)
A <- unserialize(as.raw(dataset$serialized))
```

Install new R packages

You can add R packages that are not installed by default in Azure Machine Learning. Adding new packages requires these steps:

- Obtain the Windows binaries for the package, in zipped format.
- Zip the desired package and any dependencies into a new single compressed archive file with the .ZIP extension.
- Upload the zipped file as a dataset to your workspace.
- Connect the new dataset to the **Execute R Script** module.
- Install the package using R script in a module.

The following procedure adds a new package together with its dependencies.

1. Download the zipped file for the package that you want to import to Azure Machine Learning. Be sure to get the Windows version of the zipped file.

NOTE

If you have already extracted the R package that you want to use in your workspace, you must either re-zip the package, or provide the original ZIP file when you can upload the R package to Studio (classic).

2. Check for any dependencies and if the package needs other packages that are not already in Azure ML Studio (classic), download them in zipped format and add them to the archive file.
3. Right-click the zipped file for the package you want to upload, as well as its dependencies, click **Send to**, and then select **Compressed (zipped) folder**.

TIP

The compressed folder should contain at least one zipped file with the target package, plus additional zip files containing required packages.

4. Upload the single ZIP file containing all packages (as well as any optional R code files or data files) to your Studio (classic) workspace.

You do this like you would upload a dataset: Click **New**, click **Dataset**, and then select **From local file**

and the **Zip file** option.

5. Open the **Saved Datasets** list, click **My Datasets**, and verify that the zipped file is available.
6. Drag it into your experiment, right-click the dataset, and select **Visualize** to view the files included in the zipped folder. The file names that you see in the **Contents** list are the names that you must reference when you install the package.

For example, suppose you had uploaded a file named `NewRPackage.zip`, which contains three R packages named `001.zip`, `002.zip`, and `003.zip`. In the **Datasets** list, the name of the dataset would be `NewRPackage.zip`, with contents `001.zip`, `002.zip`, and `003.zip`.

7. Connect the dataset (`NewRPackage.zip`) to the **Script Bundle** input port.

At this point, the outer zipped folder is extracted into the workspace sandbox, in the path `src`. You would now have the following packages available to you:

- `src\001.zip`
- `src\002.zip`
- `src\003.zip`

8. To install the R packages, you extract each package from its zip file, and then load the contained library.

For example, assuming the file `src\001.zip` contains the custom R package `code001`, you would run the following script:

```
# install R package contained in src\001.zip
install.packages("src/001.zip", lib = ".", repos = NULL, verbose = TRUE)
library(code001, lib.loc=".", verbose=TRUE)
```

9. Repeat the installation process for any required packages.

```
# install R package contained in src\002.zip
install.packages("src/002.zip", lib = ".", repos = NULL, verbose = TRUE)
library(code002, lib.loc=".", verbose=TRUE)
# install R package contained in src\003.zip
install.packages("src/003.zip", lib = ".", repos = NULL, verbose = TRUE)
library(code003, lib.loc=".", verbose=TRUE)
```

NOTE

If there are any dependencies among multiple packages being installed, be sure to install required packages first, or you might get an error.

Installation of all R packages must be performed as part of the experiment, to ensure that all required packages are included in the workspace that is sent to the Azure job queue when your experiment is executed.

Packages in a workspace are not persisted after the experiment has run or after you have closed your session. However, any packages that you have uploaded as zipped files can be quickly extracted and used when you re-run the experiment.

Technical notes

Optimizing R performance in Studio (classic)

The current default memory is 14 GB. You might encounter an out-of-memory error message if you attempt to manipulate very large data frames by using the **Execute R Script** module.

To increase the amount of memory that is used by R script, you can use a line similar to this at the beginning of the script:

```
memory.limit(56000)
```

User-specified R code is run by a 64-bit R interpreter that runs in Azure using an A8 virtual machine with 56 GB of RAM. To increase the speed of your R code, you can use the just-in-time compiler provided in the preinstalled **Compiler** package.

Converting data types between R and Studio (classic)

The following table shows how the data types in R correspond to the data types in Azure Machine Learning:

R TYPE	STUDIO (CLASSIC) TYPE
Integer	Integer
Double	Double
Complex	Complex This type is supported by only a subset of modules.
Logical	Boolean
Character	String
Raw	Not supported
Difftime	TimeSpan
factor	categorical
data.frame	dataset

Columns of data type `lists` in R cannot be converted because the elements in such columns potentially are of different types and sizes. For example, the following valid R code fails if used in the **Execute R Script** module:

```
data.set <- data.frame(r=I(list(list(1,2,3),list(4,5))))
maml.mapOutputPort("data.set")
```

Converting datetime values

Azure Machine Learning Studio (classic) uses different datetime types than does R. If the data that you are analyzing contains date or time data, you should be aware of the following conversion requirements when porting existing R code into Studio (classic):

Converting from Azure Machine Learning Studio (classic) to R

DateTime columns are converted to POSIXct vectors. However, each individual element of the resulting vector is a number of seconds since 1970-01-01T00:00:00.

No time zone adjustments are made in this conversion.

Converting from R to Studio (classic)

POSIXct vectors are converted to DateTime columns in the UTC time zone.

For example, 2011-03-27 01:30:00 PDT will be converted to 2011-03-27T08:30:00Z, where the Z indicates that the time is in UTC.

TIP

When using times inside the **Execute R Script** module, you must specify time stamps explicitly. The R interpreter hosted in the **Execute R Script** module does not have access to local time zone definitions.

Networking

For security reasons, all networking from or to R code in **Execute R Script** modules is blocked by Azure. Also, with very few exceptions, access to local ports from the **Execute R Script** is blocked.

Parallel execution

Currently parallel execution with multiple threads is not supported.

Expected inputs

NAME	TYPE	DESCRIPTION
<i>Dataset1</i>	Data Table	Input dataset 1
<i>Dataset2</i>	Data Table	Input dataset 2
<i>Script Bundle</i>	Zip	Set of R sources

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
R Script	Any	StreamReader		Specify a <code>StreamReader</code> that points to the R script sources.
Random Seed	≥ 0	Integer		Define a random seed value for use inside the R environment. Equivalent to <code>\\"set.seed(value)\\"</code> This parameter is optional.

Outputs

NAME	TYPE	DESCRIPTION
Result Dataset	Data Table	Output dataset

NAME	TYPE	DESCRIPTION
R Device	Data Table	Console output and PNG graphics device from the R interpreter

See also

[R Language Modules](#)

[Create R Model](#)

[Module Categories and Descriptions](#)

[Python Language Modules](#)

Create R Model

3/10/2021 • 7 minutes to read • [Edit Online](#)

Creates an R model using custom resources

Category: [Data Transformation / Manipulation](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Create R Model** module in Azure Machine Learning Studio (classic), to create an untrained model from an R script.

You can base the model on any learner that is included in an R package in the Azure Machine Learning environment.

After you create the model, you can use [Train Model](#) to train the model on a dataset, like any other learner in Azure Machine Learning. The trained model can be passed to [Score Model](#) to use the model to make predictions. The trained model can then be saved, and the scoring workflow can be published as a web service.

WARNING

Currently it is not possible to pass the scored results of an R model to [Evaluate Model](#) or [Cross-Validate Model](#). If you need to evaluate a model, you can write custom R script and run it using the [Execute R Script](#) module.

In addition to using the **Create R Model** to save and re-use custom R modules, you can create your own implementation of a modeling and data management process using R, upload the files in zipped format to your workspace, and then register the package as a custom module. For more information, see [Custom R Modules](#).

How to configure Create R Model

Use of this module requires intermediate or expert knowledge of R. The module supports use of any learner that is included in the R packages already installed in Azure Machine Learning.

This sample from the [Azure AI Gallery](#) implements a two-class Naïve Bayes classifier by using the popular `e1071` package: + [Create R Model](#). We recommend that you copy the example to your workspace and follow along.

1. Add these modules to your experiment: [Create R Model](#), [Train Model](#), [Score Model](#).
2. In the **Properties** pane of **Create R Model**, provide these scripts:
 - **Trainer R script:** The R script that you provide here is used to train the model. When you run the experiment, it is deployed to the [Train Model](#) module.
 - **Scorer R script:** The R script that you provide on this input is for scoring only. when you run the

experiment, it is deployed to the [Score Model](#) module.

3. The sample experiment also include the [Execute Python Script](#) module, which is used to plot graphs for model evaluation. This module is optional when publishing to a web service but useful when developing the experiment.

- To view the charts from the Python script, right-click the Python module, select **Python Device**, and select **Visualize**.
- To view just the model metrics, right-click the Python module, select **Python Dataset**, and select **Visualize**.

For the code in the optional Python module, see [Python module for model evaluation](#).

Training script

The following example demonstrates the type of code you might use in [Trainer R script](#).

This script loads an R package, creates model using a learner from the package, and configures the feature and label columns using the predefined constants and functions provided in [Create R Model](#).

```
library(e1071)
features <- get.feature.columns(dataset)
labels   <- as.factor(get.label.column(dataset))
train.data <- data.frame(features, labels)
feature.names <- get.feature.column.names(dataset)
names(train.data) <- c(feature.names, "Class")
model <- naiveBayes(Class ~ ., train.data)
```

- The first line loads the R package, **e1071**, which contain the Naïve Bayes classifier algorithm we want to use. Since this is one of the packages pre-installed in the Azure Machine Learning environment, you don't need to download or install the package.
- The next lines get the feature columns and the label column from the dataset, and combine them into a new R data frame that is named `train.data`:

```
features <- get.feature.columns(dataset)
labels <- as.factor(get.label.column(dataset))
train.data <- data.frame(features, labels)
feature.names <- get.feature.column.names(dataset)
```

- Note use of these predefined functions:
 - `get.label.column()` returns the column that is selected as the class label in the [Train Model](#) module.
 - `get.feature.columns()` selects the columns that were designated as features in the dataset.

By default, all columns except the label column are considered features in Studio (classic).

Therefore, to mark specific columns as features, use [Edit Metadata](#), or select a set of columns within the R script.

- `get.feature.column.names(dataset)` gets feature column names from the dataset.
- The names from the combined dataset are designated as the names for columns in `train.data`, and a temporary name `Class` is created for the label column.

```
names(train.data) <- c(feature.names, "Class")
```

- The final line of the code defines the Naïve Bayes classifier algorithm as a function of the variables

(features) and outcomes (labels) in the `train.data` data frame.

```
model <- naiveBayes(Class ~ ., train.data)
```

- Throughout the model creation, training, and scoring scripts, you must use the variable name `model`.

Scoring script

The following code illustrates the type of R code that you would provide in **Scorer R script**.

```
library(e1071)
probabilities <- predict(model, dataset, type="raw")[,2]
classes <- as.factor(as.numeric(probabilities >= 0.5))
scores <- data.frame(classes, probabilities)
```

- The first line loads the package.
- The second line computes the predicted probabilities for the scoring dataset by using the trained model from the training script, designated by the required variable name, `model`.
- The third line applies a threshold of 0.5 to probabilities when assigning the predicted class labels.
- The final line combines the class labels and probabilities into the output data frame, `scores`.
- The data frame that gets passed to the [Score Model](#) module must have the name `scores`.

Optional Python evaluation script

The sample experiment in the Azure AI Gallery includes the following Python script, which is used to generate metrics and charts for model evaluation.

```

def azureml_main(dataframe):
    import matplotlib
    matplotlib.use("agg")

    from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score, roc_curve
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt

    scores = dataframe[:, ("Class", "classes", "probabilities")]
    ytrue = scores["Class"]
    ypred = np.array([float(val) for val in scores["classes"]])
    probabilities = scores["probabilities"]

    accuracy, precision, recall, auc = \
        accuracy_score(ytrue, ypred), \
        precision_score(ytrue, ypred), \
        recall_score(ytrue, ypred), \
        roc_auc_score(ytrue, probabilities)

    metrics = pd.DataFrame();
    metrics["Metric"] = ["Accuracy", "Precision", "Recall", "AUC"];
    metrics["Value"] = [accuracy, precision, recall, auc]

# Plot ROC Curve
    fpr, tpr, thresholds = roc_curve(ytrue, probabilities)
    fig = plt.figure()
    axis = fig.gca()
    axis.plot(fpr, tpr, linewidth=8)
    axis.grid("on")
    axis.set_xlabel("False positive rate")
    axis.set_ylabel("True positive rate")
    axis.set_title("ROC Curve")
    fig.savefig("roc.png")

    return metrics,

```

Publish the custom R model workflow as a web service

After you have run the experiment, you can publish the complete experiment as a web service.

For updated instructions on how to create a web service from a Studio (classic) experiment, see [Walkthrough Step 5: Deploy the Azure Machine Learning web service](#)

By default, the web service expects all input columns from the training data to be provided, including the label column. You can add an instance of [Select Columns in Dataset](#) between the input data source and the [Score Model](#) module to exclude the label you're trying to predict.

Technical notes

- The [Create R Model](#) module supports use of CRAN R only. You cannot select another version of R, or use Microsoft R Open.
- The model is cached after the first run of the module and the module is not invoked in subsequent runs until any changes in input scripts are done. Please take this behavior into consideration if your R scripts use any of the following:
 - Functions that generate random numbers
 - Functions that generate random numbers
 - Other nondeterministic functions
- Custom R models created with this module cannot be used with these modules:

- [Tune Model Hyperparameters](#)
- [Cross-Validate Model](#)
- [One-vs-All Multiclass](#)
- [Ordinal Regression](#)
- R models do not automatically perform feature normalization of categorical data or handle missing values. Handling of such variables should be done within the training and scoring R scripts.

Table of pre-defined functions

USAGE	DESCRIPTION
<code>get.feature.columns(dataset)</code>	Gets all feature columns.
<code>get.label.column(dataset, label.type=TrueLabelType)</code>	Gets the label column, given the type. See the Constants section for a list of the available types.
<code>get.label.column.names(dataset)</code>	Gets the names of all label columns.
<code>get.label.column.name(dataset, label.type=TrueLabelType)</code>	Gets the name of the label column, given the type. See the Constants section for a list of the available types.
<code>get.label.column.types(dataset)</code>	Gets the types of all label columns.
<code>get.feature.column.names(dataset)</code>	Gets the names of all feature columns.
<code>dataset <- set.score.column(dataset, score.type, column.name)</code>	Sets the score column, given a type. See the Constants section for a list of the available types.
<code>dataset <- set.feature.channel(dataset, channel.name, column.names)</code>	Sets the feature channel, given a name. See the Constants section for a list of the available names.

Table of pre-defined constants

CONSTANT	DESCRIPTION
<code>TrueLabelType</code>	True label column type
<code>ScoredLabelType</code>	Scored label column type
<code>RawScoreType</code>	Raw score column type
<code>CalibratedScoreType</code>	Calibrated score column type
<code>ScoredProbabilitiesMulticlassColumnTypePattern</code>	The pattern to prepare scored probabilities column type for multiclass classifier
<code>BayesianLinearRegressionScoresFeatureChannel</code>	The name of the feature channel with Bayesian linear regression scores

CONSTANT	DESCRIPTION
BinaryClassificationScoresFeatureChannel	The name of the feature channel with binary classification scores
MulticlassClassificationScoresFeatureChannel	The name of the feature channel with multiclass classification scores
OrdinalRegressionScoresFeatureChannel	The name of the feature channel with ordinal regression scores
RegressionScoresFeatureChannel	The name of the feature channel with regression scores

Examples

For additional examples of how to use this module in machine learning experiments, see the [Azure AI Gallery](#).

Expected inputs

NAME	TYPE	DESCRIPTION
<i>Trainer R script</i>	Script	An R script that takes a dataset as input and outputs an untrained model.
<i>Scorer R script</i>	Script	An R script that takes a model and a dataset as input and outputs the scores specified in the script.

Outputs

NAME	TYPE	DESCRIPTION
Model	ILearner interface	An untrained model

See also

[Execute R Script](#)

[R Language Modules](#)

R Packages supported by Azure Machine Learning Studio (classic)

3/24/2021 • 15 minutes to read • [Edit Online](#)

This article lists the packages included by default in Azure Machine Learning Studio (classic). To use one of the preloaded packages in your R code, you simply import the package using standard R syntax.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

IMPORTANT

Packages available in Studio (classic) can be updated, or the version refreshed, without warning. To get the latest and most complete list of the R packages that are in the Azure Machine Learning Studio (classic) environment, we recommend that you use [this script](#).

Unsupported packages

A number of packages (not listed here) are included in the Azure Machine Learning environment but cannot be called from R code because of the following issues:

- The package has a Java dependency.
- The package binaries are not compatible with the sandboxed Azure environment.
- The package requires direct Internet access, or network access.

Use R code to return package list as dataset

To get a list of the R packages in the current environment, add the following code to an instance of the [Execute R Script](#):

```
data.set <- data.frame(installed.packages())
maml.mapOutputPort("data.set")
```

List of supported packages

Over 650 R packages are preloaded in the Azure Machine Learning environment. This section lists the packages from CRAN and Microsoft R Open that are supported as of the date of this update (10/2018).

Index

[A](#) – [B](#) – [C](#) – [D](#) – [E](#) – [F](#) – [G](#) – [H](#) – [I](#) – [J](#) – [K](#) – [L](#) – [M](#) – [N](#) – [O](#) – [P](#) – [Q](#) – [R](#) – [S](#) – [T](#) – [U](#) – [V](#) – [W](#) – [X](#) – [Y](#) – [Z](#)

The **Compatibility** column indicates whether the package is included with CRAN R 3.1 or Microsoft R Open (MRO; currently 3.2.2 and 3.4.4). If not otherwise marked, the package is included with both.

NOTE

Many new packages are now available from Microsoft R Open. This means that you can easily use the same R code in Azure ML Studio (classic), Microsoft R Server, and in SQL Server Machine Learning Services.

A

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
abc	x	x	x
abc.data		x	x
abind	x	x	x
acepack		x	x
actuar	x	x	x
ade4	x	x	x
AdMit	x	x	x
aod		x	x
ape	x	x	x
aplypack	x	x	x
approximator	x	x	x
arm	x	x	x
arules	x	x	x
arulesViz	x	x	x
ash	x	x	x
assertthat	x	x	x
AtelieR	x	x	x
AzureML		x	x

[List of supported packages](#)**B**

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
BaBooN	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
BACCO	x	x	x
backports			x
BaM	x	x	x
bark	x		
BAS	x	x	x
base	x	x	x
base64enc		x	x
BayesDA	x	x	x
bayesGARCH	x	x	x
bayesm	x	x	x
bayesmix	x	x	x
bayesQR	x	x	x
bayesSurv	x	x	x
Bayesthresh	x	x	x
BayesTree	x	x	x
BayesValidate	x	x	x
BayesX	x	x	x
BayHaz	x	x	x
bbemkr	x	x	x
BCBCSF	x	x	x
BCE	x	x	x
bclust	x	x	x
bcp	x	x	x
BenfordTests	x	x	x
bfp	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
BH	x	x	x
bindr			x
bindrcpp			x
bisoreg	x	x	x
bit	x	x	x
bit64	x	x	x
bitops	x	x	x
blob			x
BLR	x	x	x
BMA	x	x	x
Bmix	x	x	x
BMS	x	x	x
bnlearn	x	x	x
boa	x	x	x
Bolstad	x		
boot	x	x	x
bootstrap	x	x	x
bqtl	x	x	x
BradleyTerry2	x	x	x
brew	x	x	x
brglm	x	x	x
broom			x
bspec	x	x	x
bspmma	x	x	x
BVS		x	x

[List of supported packages](#)

C

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
C50	x	x	x
cairoDevice	x	x	x
calibrator	x	x	x
car	x	x	x
carData			x
caret	x	x	x
catnet	x	x	x
caTools	x	x	x
cclust		x	x
cellranger			x
checkmate			x
checkpoint			x
chron	x	x	x
class	x	x	x
classInt			x
cli			x
clue	x	x	x
cluster	x	x	x
clusterSim	x	x	x
clv		x	x
coda	x	x	x
codetools	x	x	x
coin	x	x	x
colorspace	x	x	x
combinat	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
commonmark			x
compiler	x	x	x
CORElearn	x	x	x
corpcor	x	x	x
corrplot	x	x	x
crayon		x	x
crosstalk			x
cslogistic	x	x	x
ctv	x	x	x
cubature	x	x	x
Cubist			x
curl		x	x
CVST			x
cvTools		x	x

List of supported packages

D

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
data.table	x	x	x
datasets	x	x	x
date	x	x	x
DBI	x	x	x
dclone	x	x	x
ddalpha			x
deal	x	x	x
Deducer	x	x	x
DeducerExtras	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
deepnet	x	x	x
Defaults	x		
deldir	x	x	x
dendextend		x	x
DEoptimR	x	x	x
Deriv			x
desc			x
deSolve	x	x	x
devtools	x	x	x
DiagrammeR		x	x
dichromat	x	x	x
digest	x	x	x
dimRed			x
diptest			x
distrom	x	x	x
dlm	x	x	x
DMwR		x	x
doParallel		x	x
doRSR		x	x
doSNOW	x	x	x
dotCall64			x
downloader			x
dplyr	x	x	x
DPackage	x	x	x
DRR			x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
dse	x	x	x
DT			x
dtw		x	x

List of supported packages

E

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
e1071	x	x	x
earth	x	x	x
EbayesThresh	x	x	x
ebdbNet	x	x	x
effects	x	x	x
ellipse	x	x	x
emulator	x	x	x
ensembleBMA	x	x	x
entropy	x	x	x
estimability			x
EvalEst	x	x	x
evaluate	x	x	x
evdbayes	x	x	x
exactLoglinTest	x	x	x
expint			x
expm	x	x	x
extremevalues	x	x	x

List of supported packages

F

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
FactoMineR	x	x	x
factorQR	x	x	x
faoutlier	x	x	x
fBasics	x	x	x
fields		x	x
filehash	x	x	x
fitdistrplus	x	x	x
flashClust	x	x	x
flexmix			x
FME	x	x	x
FNN			x
forcats			x
foreach	x	x	x
forecast	x	x	x
foreign	x	x	x
formatR	x	x	x
Formula	x	x	x
fpc			x
fracdiff	x	x	x
fTrading	x	x	x

List of supported packages

G

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
gam	x	x	x
gamlr	x	x	x
gbm	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
gclus	x	x	x
gdata	x	x	x
gee	x	x	x
gender		x	x
genetics	x	x	x
geoR	x	x	x
geoRglm	x	x	x
geosphere	x	x	x
GGally	x	x	x
ggdendro	x	x	x
ggformula			x
ggmap	x	x	x
ggmcmc	x	x	x
ggplot2	x	x	x
ggthemes	x	x	x
git2r		x	x
glmmBUGS	x	x	x
glmnet	x	x	x
glue			x
gmodels	x	x	x
gmp	x	x	x
gnm	x	x	x
googlePublicData	x	x	x
googleVis	x	x	x
gower			x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
GPArotation	x	x	x
gplots	x	x	x
graphics	x	x	x
grDevices	x	x	x
gregmisc	x		
grid	x	x	x
gridBase		x	x
gridExtra	x	x	x
growcurves	x	x	x
grpreg	x	x	x
gss	x	x	x
gsubfn	x	x	x
gtable	x	x	x
gtools	x	x	x
gWidgets	x	x	x
gWidgetsRGtk2	x	x	x
gWidgetscltk		x	x

List of supported packages

H

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
haplo.stats	x	x	x
hash	x	x	x
haven			x
hbssa	x	x	x
hdrcde	x	x	x
heavy	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
hexbin	x	x	x
hflights	x	x	x
HH	x	x	x
HI	x	x	x
highr	x	x	x
HistData	x	x	x
Hmisc	x	x	x
hms			x
HSAUR	x	x	x
htmlTable			x
htmltools	x	x	x
htmlwidgets		x	x
httpuv	x	x	x
httr	x	x	x

List of supported packages

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
IBrokers	x	x	x
ifultools		x	x
igraph	x	x	x
influenceR			x
inline	x	x	x
intervals	x	x	x
inum			x
iplots		x	x
ipred	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
irlba		x	x
irr	x	x	x
iterators	x	x	x

List of supported packages

J

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
JavaGD	x	x	x
JGR	x	x	x
jpeg	x	x	x
jsonlite	x	x	x

List of supported packages

K

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
kernlab	x	x	x
KernSmooth	x	x	x
KFKSDS	x	x	x
kinship2	x	x	x
kknn	x	x	x
klaR	x	x	x
knitr	x	x	x
ks	x	x	x

List of supported packages

L

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
labeling	x	x	x
labelled			x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
laeken			x
Lahman	x	x	x
lars	x	x	x
lattice	x	x	x
latticeExtra	x	x	x
lava	x	x	x
lavaan	x	x	x
lazyeval	x	x	x
leaps	x	x	x
LearnBayes	x	x	x
libcoin			x
LiblineaR	x	x	x
limSolve	x	x	x
lme4	x	x	x
lmm	x	x	x
lmPerm	x		
lmtest	x	x	x
locfit	x	x	x
locpol		x	x
LogicReg	x	x	x
longitudinalData		x	x
lpSolve	x	x	x
lsa	x	x	x
LSAfun		x	x
lubridate	x	x	x

[List of supported packages](#)

M

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
magic	x	x	x
magrittr	x	x	x
manipulateWidget			x
mapdata	x	x	x
mapproj	x	x	x
maps	x	x	x
maptools	x	x	x
maptree	x	x	x
markdown	x	x	x
MASS	x	x	x
MasterBayes	x	x	x
Matrix	x	x	x
matrixcalc	x	x	x
MatrixModels	x	x	x
maxent	x	x	x
maxLik	x	x	x
mboost	x	x	x
mclust			x
mcmc	x	x	x
MCMCglmm	x	x	x
MCMCpack	x	x	x
mda	x	x	x
memoise	x	x	x
methods	x	x	x
mgcv	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
mi		x	x
mice	x	x	x
microbenchmark	x	x	x
mime	x	x	x
miniCRAN		x	x
miniUI			x
minpack.lm	x	x	x
minqa	x	x	x
mirt		x	x
misc3d	x	x	x
miscF	x	x	x
miscTools	x	x	x
mixtools	x	x	x
mlbench	x	x	x
mlogitBMA	x	x	x
mnormt	x	x	x
MNP	x	x	x
modeest			x
ModelMetrics			x
modeltools	x	x	x
mombf	x	x	x
monomvn	x	x	x
monreg	x	x	x
mosaic	x	x	x
mosaicCore			x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
mosaicData	x	x	x
MSBVAR	x	x	x
msm	x	x	x
multcomp	x	x	x
multicool	x	x	x
munsell	x	x	x
mvoutlier	x	x	x
mvtnorm	x	x	x

[List of supported packages](#)

N

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
ncvreg	x	x	x
network			x
nlme	x	x	x
nloptr	x	x	x
NLP	x	x	x
NMF		x	x
nnet	x	x	x
nnls	x	x	x
numbers	x	x	x
numDeriv	x	x	x

[List of supported packages](#)

O

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
OceanView			x
openNLP	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
openNLPdata	x	x	x
openssl		x	x
OutlierDC	x	x	x
OutlierDM	x	x	x
outliers	x	x	x

List of supported packages

P

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
pacbpred	x	x	x
parallel	x	x	x
partitions	x	x	x
party	x	x	x
partykit		x	x
PAWL	x	x	x
pbapply			x
pbivnorm	x	x	x
pbkrtest			x
pcaPP	x	x	x
pdc		x	x
PerformanceAnalytics	x	x	x
permute	x	x	x
pillar			x
pkgconfig			x
pkgmaker		x	x
pkgXMLBuilder		x	x
plogr			x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
plot3D			x
plot3Drgl			x
plotly			x
plotmo	x	x	x
plotrix	x	x	x
pls	x	x	x
plyr	x	x	x
png	x	x	x
polyspline	x	x	x
polynom	x	x	x
PottsUtils	x	x	x
prabclus			x
praise		x	x
predmixcor	x	x	x
PresenceAbsence	x	x	x
prettyunits			x
pROC	x	x	x
prodlm	x	x	x
profddpm	x	x	x
profileModel	x	x	x
progress			x
proto	x	x	x
proxy		x	x
pryr	x	x	x
pscl	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
psych	x	x	x
purr			x

List of supported packages

Q

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
qap		x	x
qdap		x	x
qdapDictionaries	x	x	x
qdapRegex	x	x	x
qdapTools	x	x	x
quadprog	x	x	x
quantmod	x	x	x
quantreg	x	x	x
questionr			x
qvcalc	x	x	x

List of supported packages

R

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
R.matlab	x	x	x
R.methodsS3	x	x	x
R.oo	x	x	x
R.utils	x	x	x
R2HTML	x	x	x
R2jags	x	x	x
R2OpenBUGS	x	x	x
R2WinBUGS	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
R6	x	x	x
ramps	x	x	x
RandomFields	x	x	x
RandomFieldsUtils		x	x
randomForest	x	x	x
RArcInfo	x	x	x
raster	x	x	x
rattle	x	x	x
rbenchmark	x	x	x
rbugs	x	x	x
RColorBrewer	x	x	x
Rcpp	x	x	x
RcppArmadillo	x	x	x
rcppbugs	x	x	x
RcppEigen	x	x	x
RcppExamples	x	x	x
RcppRoll			x
RCurl	x	x	x
readr		x	x
readxl			x
recipes			x
registry		x	x
relimp	x	x	x
rematch			x
reports	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
reshape	x	x	x
reshape2	x	x	x
RevoIOQ		x	x
revolpe		x	x
RevoMods		x	x
RevoPemaR		x	x
RevoRpeConnector		x	x
RevoRsrConnector		x	x
RevoTreeView		x	x
RevoUtils		x	x
RevoUtilsMath		x	x
rgdal	x	x	x
rgeos	x	x	x
regx			x
rgl		x	x
RgoogleMaps	x	x	x
RGraphics	x	x	x
RGtk2	x	x	x
RINside		x	x
RJaCGH	x	x	x
rjags		x	x
Rjava	x	x	x
rjson	x	x	x
RJSONIO	x	x	x
rlang			x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
rlecuyer		x	x
Rmpfr		x	x
rms		x	x
RMySQL		x	x
rngtools		x	x
robCompositions	x	x	x
robustbase	x	x	x
ROCR	x	x	x
RODBC	x	x	x
Rook			x
rootSolve	x	x	x
roxygen	x		
roxygen2	x	x	x
rpart	x	x	x
rpart.plot	x	x	x
rprojroot			x
rrcov	x	x	x
rscproxy	x	x	x
RSGHB	x	x	x
RSNNS	x	x	x
RSQLite	x	x	x
rstudioapi		x	x
RTextTools	x	x	x
RUnit	x	x	x
runjags	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
Runuran	x	x	x
RWekajars	x	x	x
rworldmap	x	x	x
rworldxtra	x	x	x

List of supported packages

S

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
SampleSizeMeans	x	x	x
SampleSizeProportions	x	x	x
sandwich	x	x	x
sbgcop	x	x	x
scales	x	x	x
scapeMCMC	x		
scatterplot3d	x	x	x
sciplot	x	x	x
segmented	x	x	x
sem	x	x	x
seriation	x	x	x
setRNG	x	x	x
sfsmisc		x	x
sgeostat	x	x	x
shape			x
shapefiles	x	x	x
shiny	x	x	x
SimpleTable	x	x	x
SIS	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
skmeans	x	x	x
slam	x	x	x
smoothSurv	x	x	x
sna	x	x	x
snow	x	x	x
SnowballC	x	x	x
snowFT	x	x	x
sourcetools			x
sp	x	x	x
spacetime	x	x	x
spam		x	x
SparseM	x	x	x
spatial	x	x	x
spBayes	x	x	x
spData			x
spdep	x	x	x
spikeslab	x	x	x
splancs	x	x	x
splines	x	x	x
spls	x	x	x
splus2R		x	x
spTimer	x	x	x
SQUAREM			x
sqldf	x	x	x
sROC		x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
stabledist	x	x	x
stabs	x	x	x
statnet.common			x
stats	x	x	x
stats4	x	x	x
stepPlr	x	x	x
stochvol	x	x	x
stringdist	x	x	x
stringi	x	x	x
stringr	x	x	x
strucchange	x	x	x
stsm	x	x	x
stsm.class	x	x	x
SuppDists	x	x	x
survey			x
survival	x	x	x
svmpath	x	x	x
svUnit	x	x	x

List of supported packages

T

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
tau	x	x	x
tcltk	x	x	x
tcltk2	x	x	x
TeachingDemos	x	x	x
tensorA	x	x	x

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
testthat	x	x	x
textcat	x	x	x
textr	x	x	x
tfplot	x	x	x
tframe	x	x	x
tgp	x	x	x
TH.data	x	x	x
tibble			x
tidyrr	x	x	x
tidyselect			x
timeDate	x	x	x
timeSeries	x	x	x
tm	x	x	x
tools	x	x	x
topicmodels	x	x	x
tree	x	x	x
trimcluster			x
TSclust		x	x
tseries	x	x	x
tsfa	x	x	x
tsoutliers	x	x	x
TSP	x	x	x
TTR	x	x	x
twitteR	x	x	x

List of supported packages

U

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
UsingR	x	x	x
utf8			x
utils	x	x	x
uuid		x	x

[List of supported packages](#)

V

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
varSelectIP	x		
vcd	x	x	x
vegan	x	x	x
venneuler	x	x	x
VGAM	x	x	x
VIF	x	x	x
VIM		x	x
viridis			x
viridisLite			x
visNetwork		x	x

[List of supported packages](#)

W

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
wavethresh			x
whisker	x	x	x
withr		x	x
wmtsa		x	x
wordcloud	x	x	x

[List of supported packages](#)

X

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
xgboost		x	x
XLConnect	x	x	x
XLConnectJars	x	x	x
xlsx	x	x	x
xlsxjars	x	x	x
XML	x	x	x
xml2			x
xtable	x	x	x
xts	x	x	x

List of supported packages

Y

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
yaml	x	x	x

List of supported packages

Z

PACKAGE NAME	CRAN R 3.1	MRO 3.2.2	MRO 3.4.4
zic	x	x	x
zipfR	x	x	x
zoo	x	x	x

List of supported packages

See also

[R Language Modules](#)

Statistical Functions

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that support mathematical and statistical operations critical for machine learning. If you need to perform tasks such as the following in your experiment, look in the **Statistical Functions** category:

- Perform ad hoc computations on column values, such as rounding or using an absolute value.
- Compute means, logarithms, and other statistics commonly used in machine learning.
- Calculate correlation and probability scores.
- Compute z-scores.
- Compute widely used statistical distributions, such as Weibull, gamma, and beta.
- Generate statistical reports over a set of columns or a dataset.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

For example, if you have a new dataset, you might use the [Summarize Data](#) module first. It generates a report for an entire dataset that includes standard statistical measures, such as mean and standard deviation.

If you need more advanced statistics, such as sample skewness or interquartile distance, use the [Compute Elementary Statistics](#) module to generate additional descriptive statistics.

Because the modules generate the results each time you run the experiment, the results are updated if your data changes.

List of modules

The **Statistical Functions** category includes the following modules:

- [Apply Math Operation](#): Applies a mathematical operation to column values.
- [Compute Elementary Statistics](#): Calculates specified summary statistics for selected dataset columns.
- [Compute Linear Correlation](#): Calculates the linear correlation between column values in a dataset.
- [Evaluate Probability Function](#): Fits a specified probability distribution function to a dataset.
- [Replace Discrete Values](#): Replaces discrete values from one column with numeric values based on another column.
- [Summarize Data](#): Generates a basic descriptive statistics report for the columns in a dataset.
- [Test Hypothesis Using t-Test](#): Compares means from two datasets by using a t-test.

See also

- [Scale and Reduce module](#)
- [Feature Selection category](#)
- [Learning with Counts module](#)
- [Module categories and descriptions](#)

- [A-Z module list](#)

Apply Math Operation

3/10/2021 • 19 minutes to read • [Edit Online](#)

Applies a mathematical operation to column values

Category: [Statistical Functions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Apply Math Operation** module in Azure Machine Learning Studio (classic), to create calculations that are applied to numeric columns in the input dataset.

Supported math operations include common arithmetic functions such as multiplication and division, trigonometric functions, a variety of rounding functions, and special functions used in data science such as gamma and error functions.

After you define an operation and run the experiment, the values are added to your dataset. Depending on how you configure the module, you can:

- Append the results to your dataset. This is particularly useful when you are verifying the result of the operation.
- Replace columns values with the new, computed values.
- Generate a new column for results, and not show the orginal data.

TIP

This module performs a single math operation at a time. For complex math operations, we recommend using these modules instead:

- [Execute R Script](#)
- [Execute Python Script](#)
- [Apply SQL Transformation](#)

Look for the operation you need in these categories:

- [Basic](#)

The functions in the **Basic** category can be used to manipulate a single value or column of values. For example, you might get the absolute value of all numbers in a column, or calculate the square root of each value in a column.

- [Compare](#)

The functions in the **Compare** category are all used for comparison: you can do a pair-wise comparison of the values in two columns, or you can compare each value in a column to a specified constant. For

example, you could compare columns to determine whether values were the same in two datasets. Or, you might use a constant, such as a maximum allowed value, to find outliers in a numeric column.

- [Operations](#)

This category includes the basic mathematical functions: addition, subtraction, multiplication, and division. You can work with either columns or constants. For example, you might add the value in Column A to the value in Column B. Or, you might subtract a constant, such as a previously calculated mean, from each value in Column A.

- [Rounding](#)

This category includes a variety of functions for performing operations such as rounding, ceiling, floor, and truncation to various levels of precision. You can specify the level of precision for both decimal and whole numbers.

- [Special mathematical functions](#)

The **Special** category includes mathematical functions that are especially used in data science, such as elliptic integrals and the Gaussian error function.

- [Trigonometric functions](#)

This category includes all standard trigonometric functions. For example, you can convert radians to degrees, or compute functions such as tangent in either radians or degrees. These functions are unary, meaning that they take a single column of values as input, apply the trigonometric function, and return a column of values as the result. Therefore you need to make sure that the input column is the appropriate type and contains the right kind of values for the specified operation.

Examples

For examples of how to use **Apply Math Operation**, see these sample experiments in the [Azure AI Gallery](#):

- [Color quantization](#): One set of column values is subtracted from another, and then the results are squared.
- [Customer relationship prediction](#): The constant 1 is added to all values in a column to distinguish between zeros and missing values.
- [Flight delay prediction](#): Demonstrates various operations, including rounding and division.
- [Direct marketing](#): Uses comparison operations to determine whether probability scores meet a required value.

How to use Apply Math Operation

The **Apply Math Operation** module requires a dataset that contains at least one column containing only numbers. The numbers can be discrete or continuous but must be of a numeric data type, not a string.

You can apply the same operation to multiple numeric columns, but all columns must be in the same dataset.

Each instance of this module can perform only one type of operation at a time. To perform complex math operations, you might need to chain together several instances of the **Apply Math Operation** module.

1. Add the **Apply Math Operation** module to your experiment. You can find this module in the [Statistical Functions](#) category.
2. Connect a dataset that contains at least one numeric column.
3. Click **Category** to select the **type** of math operation to perform.

For example, to do basic arithmetic on columns, choose **Operations**. To get a logarithm or a ceiling, choose **Basic**. To compare columns of values, use **Comparison**.

TIP

All other options change depending on what type of mathematical operation you choose. Also, any change to the category resets all other options. Therefore, be sure to select from **Category** first!

4. Choose a specific operation from the list in that category.
5. Select one or more source columns on which to perform the calculation.
 - Any column that you choose must be a numeric data type.
 - The range of data must be valid for the selected mathematical operation. Otherwise an error or NaN (not a number) result may occur. For example, $\text{Ln}(-1.0)$ is an invalid operation and results in a value of **Nan**.
6. Set additional parameters required by each type of operation.
7. Use the **Output mode** option to indicate how you want the math operation to be generated:
 - **Append**. All the columns used as inputs are included in the output dataset, plus one additional column is appended that contains the results of the math operation.
 - **Inplace**. The values in the columns used as inputs are replaced with the new calculated values.
 - **ResultOnly**. A single column is returned containing the results of the math operation.
8. Run the experiment, or right-click just the **Apply Math Operation** module and select **Run selected**.

Results

If you generate the results using the **Append** or **ResultOnly** options, the column headings of the returned dataset indicate the operation and the columns that were used. For example, if you compare two columns using the **Equals** operator, the results would look like this:

- **Equals(Col2_Col1)**, indicating that you tested Col2 against Col1.
- **Equals(Col2_\$10)**, indicating that you compared column 2 to the constant 10.

Even if you use the **Inplace** option, the source data is not deleted or changed; the column in the original dataset is still available in Studio (classic). To view the original data, you can connect the **Add Columns** module and join it to the output of **Apply Math Operation**.

Basic math operations

The functions in the **Basic** category usually take a single value from a column, perform the predefined operation, and return a single value. For some functions you can specify a constant as a second argument.

Azure Machine Learning supports the following functions in the **Basic** category:

Abs

Returns the absolute value of the selected columns.

Atan2

Returns a four-quadrant inverse tangent.

Select the columns that contain the point coordinates. For the second argument, which corresponds to the x-coordinate, you can also specify a constant.

Corresponds to the ATAN2 function in Matlab.

Conj

Returns the conjugate for the values in the selected column.

CubeRoot

Calculates the cube root for the values in the selected column.

DoubleFactorial**

Calculates the double factorial for values in the selected column. The double factorial is an extension of the normal factorial function, and it is denoted as $x!!$.

Eps

Returns the size of the gap between the current value and the next-highest, double-precision number.

Corresponds to the EPS function in Matlab.

Exp

Returns e raised to the power of the value in the selected column. This is the same as the Excel EXP function.

Exp2

Returns the base-2 exponential of the arguments, solving for $y = x * 2^t$ where t is a column of values containing exponents.

For Exp2 you can specify a second argument x, which can be either a constant or another column of values

In **Second argument type**, indicate whether you will provide the multiplier t as a constant, or a value in a column.

You can select a single column with the exponent values, or type the exponent value in the **Constant second argument** text box. Then, in **Column set**, select the column that contains the exponent values.

For example, if you select a column with the values {0,1,2,3,4,5} for both the multiplier and the exponent, the function returns {0, 2, 8, 24, 64 160).

ExpMinus1

Returns the negative exponent for values in the selected column.

Factorial

Returns the factorial for values in the selected column.

Hypotenuse

Calculates the hypotenuse for a triangle in which the length of one side is specified as a column of values, and the length of the second side is specified either as a constant or as two columns.

ImaginaryPart

Returns the imaginary part of the values in the selected column.

Ln

Returns the natural logarithm for the values in the selected column.

LnPlus1

Returns the natural logarithm plus one for the values in the selected column.

Log

Returns the log of the values in the selected column, given the specified base.

You can specify the base (the second argument) either as a constant or by selecting another column of values.

Log10

Returns the base 10 logarithm the values in the selected column.

Log2

Returns the base 2 logarithm for the values in the selected column.

NthRoot

Returns the nth root of the value, using an n that you specify.

Select the columns for which you want to calculate the root, by using the **ColumnSet** option.

In **Second argument type**, select another column that contains the root, or specify a constant to use as the root.

If the second argument is a column, each value in the column is used as the value of n for the corresponding row. If the second argument is a constant, type the value for n in the **Constant second argument** text box.

Pow

Calculates X raised to the power of Y for each of the values in the selected column.

First, select the columns that contains the **base**, which should be a float, by using the **ColumnSet** option.

In **Second argument type**, select the column that contains the exponent, or specify a constant to use as the exponent.

If the second argument is a column, each value in the column is used as the exponent for the corresponding row. If the second argument is a constant, type the value for the exponent in the **Constant second argument** text box.

RealPart

Returns the real part of the values in the selected column.

Sqrt

Returns the square root of the values in the selected column.

SqrtPi

For each value in the selected column, multiplies the value by pi and then returns the square root of the result.

Square

Squares the values in the selected column.

Comparison operations

Use the comparison functions in Azure Machine Learning Studio (classic) any time that you need to test two sets of values against each other. For example, in an experiment you might need to do these comparison operations:

- Evaluate a column of probability scores model against a threshold value.
- Determine if two sets of results are the same, and for each row that is different, add a FALSE flag that can be used for further processing or filtering.

EqualTo

Returns True if the values are the same.

GreaterThan

Returns True if the values in **Column set** are greater than the specified constant, or greater than the corresponding values in the comparison column.

GreaterThanOrEqualTo

Returns True if the values in **Column set** are greater than or equal to the specified constant, or greater than or equal to the corresponding values in the comparison column.

LessThan

Returns True if the values in **Column set** are less than the specified constant, or less than the corresponding values in the comparison column.

LessThanOrEqualTo

Returns True if the values in **Column set** are less than or equal to the specified constant, or less than or equal to the corresponding values in the comparison column.

NotEqualTo

Returns True if the values in **Column set** are not equal to the constant or comparison column, and returns False if they are equal.

PairMax

Returns the value that is greater—the value in **Column set** or the value in the constant or comparison column.

PairMin

Returns the value that is lesser—the value in **Column set** or the value in the constant or comparison column

Arithmetic operations

Includes the basic arithmetic operations: addition and subtraction, division and multiplication. Because most operations are binary, requiring two numbers, you first choose the operation, and then choose the column or numbers to use in the first and second arguments.

The order in which you choose the columns for division and subtraction might seem counterintuitive; however, to make it easier to understand the results, the column heading provides the operation name, and the order in which the columns were used.

OPERATION	NUM1	NUM2	RESULT COLUMN	RESULT VALUE
Addition	1	5	Add(Num2_Num1)	4
Multiplication	1	5	Multiple(Num2_Num1)	5
Subtraction	1	5	Subtract(Num2_Num1)	4
Subtraction	0	1	Subtract(Num2_Num1)	0
Division	1	5	Divide(Num2_Num1)	5
Division	0	1	Divide(Num2_Num1)	Infinity

Add

Specify the source columns by using **Column set**, and then add to those values a number specified in **Constant operation argument**.

To add the values in two columns, choose a column or columns using **Column set**, and then choose a second column using **Operation argument**.

Divide

Divides the values in **Column set** by a constant or by the column values defined in **Operation argument**. In other words, you pick the divisor first, and then the dividend. The output value is the quotient.

Multiply

Multiplies the values in **Column set** by the specified constant or column values.

Subtract

Specify the number to subtract (the *subtrahend*) by using the **Operation argument** dropdown list. You can choose either a constant or column of values. Then, specify the column of values to operate on (the *minuend*), by choosing a different column, using the second **Column set** option.

You can subtract a constant from each value in a column of values, but not the reverse operation. To do this, use addition instead.

Rounding operations

Studio (classic) supports a variety of rounding operations. For many operations, you must specify the amount of precision to use when rounding. You can use either a static precision level, specified as a constant, or you can apply a dynamic precision value obtained from a column of values.

- If you use a constant, set **Precision Type** to **Constant** and then type the number of digits as an integer in the **Constant Precision** text box. If you type a non-integer, the module does not raise an error, but results can be unexpected.
- To use a different precision value for each row in your dataset, set **Precision Type** to **ColumnSet**, and then choose the column that contains appropriate precision values.

Ceiling

Returns the ceiling for the values in **Column set**.

CeilingPower2

Returns the squared ceiling for the values in **Column set**.

Floor

Returns the floor for the values in **Column set**, to the specified precision.

Mod

Returns the fractional part of the values in **Column set**, to the specified precision.

Quotient

Returns the fractional part of the values in **Column set**, to the specified precision.

Remainder

Returns the remainder for the values in **Column set**.

RoundDigits

Returns the values in **Column set**, rounded by the 4/5 rule to the specified number of digits.

RoundDown

Returns the values in **Column set**, rounded down to the specified number of digits.

RoundUp

Returns the values in **Column set**, rounded up to the specified number of digits.

ToEven

Returns the values in **Column set**, rounded to the nearest whole, even number.

ToOdd

Returns the values in **Column set**, rounded to the nearest whole, odd number.

Truncate

Truncates the values in **Column set** by removing all digits not allowed by the specified precision.

Special math functions

This category includes specialized mathematical functions often used in data science. Unless otherwise noted, the function is unary and returns the specified calculation for each value in the selected column or columns.

Beta

Returns the value of Euler's beta function.

EllipticIntegralE

Returns the value of the incomplete elliptic integral.

EllipticIntegralK

Returns the value of the complete elliptic integral (K).

Erf

Returns the value of the error function.

The error function (also called the Gauss error function) is a special function of the sigmoid shape that is used in probability to describe diffusion.

Erfc

Returns the value of the complementary error function.

Erfc is defined as $1 - \text{erf}(x)$.

ErfScaled

Returns the value of the scaled error function.

The scaled version of the error function can be used to avoid arithmetic underflow.

ErfInverse

Returns the value of the inverse erf function.

ExponentialIntegralEi

Returns the value of the exponential integral Ei.

Gamma

Returns the value of the gamma function.

GammaLn

Returns the natural logarithm of the gamma function.

GammaRegularizedP

Returns the value of the regularized incomplete gamma function.

This function takes a second argument, which can be provided either as a constant or a column of values.

GammaRegularizedPIverse

Returns the value of the inverse regularized incomplete gamma function.

This function takes a second argument, which can be provided either as a constant or a column of values.

GammaRegularizedQ

Returns the value of the regularized incomplete gamma function.

This function takes a second argument, which can be provided either as a constant or a column of values.

GammaRegularizedQInverse

Returns the value of the inverse generalized regularized incomplete gamma function.

This function takes a second argument, which can be provided either as a constant or a column of values.

PolyGamma

Returns the value of the polygamma function.

This function takes a second argument, which can be provided either as a constant or a column of values.

Trigonometric functions

This category includes most of the important trigonometric and inverse trigonometric functions. All trigonometric functions are unary and require no additional arguments.

Acos

Calculates the arccosine for the column values.

AcosDegree

Calculates the arccosine of the column values, in degrees.

Acosh

Calculates the hyperbolic arccosine of the column values.

Acot

Calculates the arccotangent of the column values.

AcotDegrees

Calculates the arccotangent of the column values, in degrees.

Acoth

Calculates the hyperbolic arccotangent of the column values.

Acsc

Calculates the arccosecant of the column values.

AcscDegrees

Calculates the arccosecant of the column values, in degrees.

Asec

Calculates the arcsecant of the column values.

AsecDegrees

Calculates the arcsecant of the column values, in degrees.

Asech

Calculates the hyperbolic arcsecant of the column values.

Asin

Calculates the arcsine of the column values.

AsinDegrees

Calculates the arcsine of the column values, in degrees.

Asinh

Calculates the hyperbolic arcsine for the column values.

Atan

Calculates the arctangent of the column values.

AtanDegrees

Calculates the arctangent of the column values, in degrees.

Atanh

Calculates the hyperbolic arctangent of the column values.

Cis

Returns a complex-valued function made from sine and cosine with the definition $\text{cis } \theta = \cos \theta + i\sin \theta$.

Cos

Calculates the cosine of the column values.

CosDegrees

Calculates the cosine for the column values, in degrees.

Cosh

Calculates the hyperbolic cosine for the column values.

Cot

Calculates the cotangent for the column values.

CotDegrees

Calculates the cotangent for the column values, in degrees.

Coth

Calculates the hyperbolic cotangent for the column values.

Csc

Calculates the cosecant for the column values.

CscDegrees

Calculates the cosecant for the column values, in degrees.

Csch

Calculates the hyperbolic cosecant for the column values.

DegreesToRadians

Converts degrees to radians.

Sec

Calculates the secant of the column values.

aSecDegrees

Calculates the secant for the column values, in degrees.

aSech

Calculates the hyperbolic secant of the column values.

Sign

Returns the sign of the column values.

Sin

Calculates the sine of the column values.

Sinc

Calculates the sine-cosine value of the column values.

SinDegrees

Calculates the sine for the column values, in degrees.

Sinh

Calculates the hyperbolic sine of the column values.

Tan

Calculates the tangent of the column values.

TanDegrees

Calculates the tangent for the argument, in degrees.

Tanh

Calculates the hyperbolic tangent of the column values.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Operations on multiple columns

Be careful when you select more than one column as the second operator. The results are easy to understand if the operation is simple, such as adding a constant to all columns.

Assume your dataset has multiple columns, and you add the dataset to itself. In the results, each column is added to itself, as follows:

NUM1	NUM2	NUM3	ADD(NUM1_NUM1)	ADD(NUM2_NUM2)	ADD(NUM3_NUM3)
1	5	2	2	10	4
2	3	-1	4	6	-2
0	1	-1	0	2	-2

If you need to perform more complex calculations, you can chain multiple instances of **Apply Math Operation**. For example, you might add two columns by using one instance of **Apply Math Operation**, and then use another instance of **Apply Math Operation** to divide the sum by a constant to obtain the mean.

Alternatively, use one of the following modules to do all the calculations at once, using SQL, R, or Python script :

- [Execute R Script](#)
- [Execute Python Script](#)
- [Apply SQL Transformation](#)

Unary and binary functions

In a *unary operation*, you create calculations based on column values without referring to other columns or

constants.

For example, you might truncate the column's values to a certain degree of precision, round values up or down, or find ceiling or floor values.

An example of a unary operation is `Abs(X)`, where X is the column that is provided as input.

In a *binary operation*, you specify two sets of values. The first argument must always be a column or set of columns, while the second argument can be a number you specify as a constant, or another column.

An example of a binary operation that uses two columns is `Subtract(X,Y)`, in which X is the first column you select, and Y is the second column.

An example of using a binary operation that combines a column and a constant might be `Subtract(X,mean)`, where you type the column mean as a constant and subtract it from each value in column X.

Handling of numbers in categorical columns

Support for categorical values presented as numbers depends on the function, and on how many arguments the function takes.

- If your operation includes numbers designated as categorical columns, a unary operation can be applied to categorical data values.
- If a unary operation is applied to a categorical column, the categorical data values of the input column can be transformed to equal associated categorical data values of the output column. In this case, the values are merged, such that the number of categorical data values in the output is always less than the number of values in the input.
- If a binary operation is applied to a categorical column and some other column, the expected behavior is as follows:
 - If the other column is dense, the output column is categorical.

Categorical data values presented in the input are lost.

The output column has only those values that are present in the output column data.

- If the other column is sparse, the output column is sparse.
- If both arguments of a binary operation are sparse columns, the resulting column contains background zeros in all positions where both input columns contained background zeros.

Processing of sparse columns

In unary operations, all elements of sparse columns that correspond to background zeros are left unprocessed.

In binary operations, if one argument is a sparse column and the other argument is a dense column, the resulting column is sparse with all background zeros propagated from input from the sparse column.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Results dataset

See also

[Statistical Functions](#)

[A-Z Module List](#)

Compute Elementary Statistics

3/10/2021 • 10 minutes to read • [Edit Online](#)

Calculates specified summary statistics for selected dataset columns

Category: [Statistical Functions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Compute Elementary Statistics](#) module in Azure Machine Learning Studio (classic), to generate a summary report for your dataset that lists key statistics such as mean, standard deviation, and the range of values for each of the selected columns.

This report is useful for analyzing the central tendency, dispersion, and shape of data.

How to configure Compute Elementary Statistics

1. Add the [Compute Elementary Statistics](#) module to your experiment. You can find this module in the [Statistical Functions](#) category in Azure Machine Learning Studio (classic).
2. Connect a dataset that contains the columns you want to analyze.
3. Click the **Method** dropdown list, and choose the type of value that you want to calculate for each column.

See the [Supported Statistics](#) section for a full list of available statistics and what they mean.

4. By default, the value you selected in **Method** dropdown list will be calculated for all columns in the dataset that have a numeric data type. If any column has values that prevent the value from being calculated, an error will be raised and the report will not be created.

To avoid this error, use the column selector to pick the numeric columns for which you want a report. All columns that you choose must be numeric.

5. Run the experiment.

Results

The generated report includes the name of each column and the statistic that was calculated. For example, the following table shows statistics generated for the **mpg** column.

DEVIATIONSQUARED(MPG)	MAX(MPG)	MIN(MPG)
9674.312	25.21951	13

TIP

Each time you run [Compute Elementary Statistics](#), it can generate only a single summary statistic for each of the selected columns. However, you can use the [Add Columns](#) or [Add Rows](#) modules to merge the results into a single table, as in the preceding example.

Supported statistics

This module supports the following standard descriptive statistics.

Deviation squared

Calculates the *squared deviation* of the column values. Also known as the sum of squares.

Squared deviation is a measure of how far values are dispersed from the mean.

Geometric mean

Calculates the *geometric mean* of the column values.

The geometric mean can be used to measure the central tendency of a set of numbers. Compared to the arithmetic mean, it is less affected by a small number of extreme values. It can also be used to compare measurements on different scales, since it effectively normalizes the scales of the numbers being compared. The geometric mean is sometimes used to estimate compound annual growth rates.

The equivalent function in Excel is GEOMEAN.

Harmonic mean

Calculates the *harmonic mean* of the column values.

To compute the harmonic mean, all values are converted to their reciprocals, and then the mean is taken of those values. The harmonic mean is the reciprocal of that mean. If the column values are positive, larger numbers are weighted less than smaller numbers.

The harmonic mean is always less than the geometric mean, which is always less than the arithmetic mean. The harmonic mean is useful for averaging variables that represent rates, such as speed (distance over time) or sales per quarter.

The equivalent function in Excel is HARMEAN.

Interquartile distance

Calculates the *interquartile difference* for the first and the last *quartiles* of the column values. Also called the *quartile range*. When the quartile falls between two numbers, the quartile value is the average of the two values on either side of the cut.

The quartile value divides the column of values into four groups with an equal number of values. Thus, one quarter of the values are less than or equal to the 25th percentile. Three quarters of the values are less than or equal to the 75th percentile. By reviewing the quartile range you can get an idea of how widely spread the data values are.

K-th central moment

Calculates *K-th central moment* for the column values.

When calculating K-th central moment, you must also specify the **Order**, meaning the value of k. The value of k can range from 0 to any allowed integer value, though higher order values are generally not meaningful.

Generally, in descriptive statistics, a moment is a measure that describes the shape of a set of points. Central moments are moments about the mean, which are usually used because they provide better information about the distribution's shape. An order of 2 usually represents the variance; an order of 4 is used for kurtosis. The first

order moment is the mean. Thus the collection of all moments uniquely describes the distribution of values in the column.

Max

Finds the *maximum value* in the column.

Mean

Calculates the *arithmetic mean* of the column values.

The equivalent function in Excel is AVERAGE.

Mean deviation

Calculates the *mean absolute deviation* for the column values.

That is, the mean is computed for the column, and the deviation computed for each value in the column. The average of the absolute values of the individual deviation values is the mean deviation.

This statistic tells you how spread out from the mean your column of numbers is.

Median

Returns the *median* of the column values.

The median is the number in the middle of a column of numbers. If there is an even number of numbers in the column, the median is the average of the two numbers in the middle.

The median, together with the *mean* and the *mode*, is one of three statistics that measures central tendency. If the values are symmetrical around the mean, the three numbers will be about the same. However, the median is more robust to outliers than the mean.

Median deviation

Calculates the *median deviation* for the column.

That is, the median is computed for the column, and the deviation computed for each value in the column. The median value of the absolute values of the individual deviation values is taken.

The median absolute deviation is also known as MAD, and is used to describe the variability of a sample of numbers. MAD tells you how spread out from the mean your column of numbers is.

Min

Returns the *minimum value* of the column values.

Mode

Finds all *modes* for the column.

The mode is the value that appears the most in the column. If several values appear the same number of times, the column can have multiple modes.

As a measure of central tendency, mode is more robust to outliers than the mean, and can be used with nominal data too.

Population standard deviation

Calculates the *population standard deviation* for the column values.

This statistic assumes that the column values represent the entire population. If your data is only a sample of the population, you must compute the standard deviation by using **Sample standard deviation**. However, in large datasets, the two statistics return approximately equal values.

The standard deviation is computed as the square root of the column variance. This statistic captures the amount of variability in the column.

Population variance

Calculates the *population variance* for the column values.

Variance measures how much a set of numbers is spread out. If variance is zero, all numbers are the same.

This statistic assumes that the column of values represents the entire population. If your data contains only a sample of the values, you should compute variance by using **Sample variance**.

The equivalent Excel function is `VAR.P`.

Product

Calculates the *product* of the column's elements.

To get the product, you multiple all the numbers in the column. The result is not in itself useful as a descriptive statistic but the function is useful for a variety of other calculations.

Range

Calculates the *range* of the column values. The range is defined as the maximum value minus the minimum value

Sample kurtosis

Calculates the *sample kurtosis* for the column values.

Kurtosis describes the shape of the distribution of values-- that is, how peaked or flat the distribution of values is, compared with the normal distribution.

- The normal distribution has a kurtosis of 0.
- High kurtosis values indicate that the probability mass is concentrated either around a peak, or in the tail of the distribution.
- Negative kurtosis values indicate a relatively flat distribution.

Sample skewness

Calculates the *sample skewness* for the column values.

Skew describes whether the bulk of the values are at the center, shifted to the left, or shifted to the right. Two distributions might have the same mean and standard deviation, yet be shaped very differently. You can use skewness and kurtosis to characterize the shape.

- Negative skew values means the distribution is skewed to the left.
- 0 denotes the normal distribution.
- Positive skewness values mean the distribution is skewed to the right.

Sample standard deviation

Calculates the *sample standard deviation* for the column values.

The standard deviation of the sample measures how spread out the values in the column are from the mean. It represents the average distance between the values of the data in the set and the mean.

This statistic assumes that the column values represent a sample of the population. If your data represents the entire population, you must compute the standard deviation using **Population standard deviation**.

The equivalent Excel function is `ST.DEVS`.

Sample variance

Calculates the *sample variance* for the column values.

This method assumes that the column values represent a sample of the population. If the column contains the entire population, you should use [Population standard variance](#).

The equivalent Excel function is VAR.S.

Sum

Calculates the *sum* of the column values.

Examples

The following experiments in the [Azure AI Gallery](#) demonstrate how you can create a summary report that contains descriptive statistics for an entire dataset. The summary report contains only general statistics; however, you can save it as a dataset and then add more detailed statistics, using the options in [Compute Elementary Statistics](#).

- [Download dataset from UCI](#): The [Summarize Data](#) module is used to generate a summary report on all columns in the dataset.
- [Dataset Processing and Analysis](#): The [Summarize Data](#) module is used to generate a summary report on all columns in the dataset.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

TIP

The following conditions must be satisfied when using the [Compute Elementary Statistics](#) module:

- There must be a sufficient number of data points (rows) to compute the selected statistic. For example, to compute [Sample standard deviation](#) requires at least two data points; otherwise, the result is NaN.
- Input columns must be numeric or Boolean.

By default, all numeric columns are selected. However, if any numeric columns are marked as categorical, you might get the following error: " Error 0056: Column with name <column name> is not in an allowed category." To correct the error, add an instance of the [Edit Metadata](#) module, select the column with the problem, and use the option [Remove categorical](#).

Implementation details

Boolean columns are processed as follows:

- MIN is computed as logical AND.
- MAX is computed as logical OR.
- RANGE checks whether the number of unique values in the column equals 2.
- Missing values are ignored.
- For statistics that require floating-point calculations, True = 1.0 and False = 0.0

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Method	List	Elementary statistics method		Selects a statistical method to use in calculations. See How to use section for list of values.
Column set	any	ColumnSelection	NumericAll	Selects the columns for which to calculate the statistic
Order	>=1	Integer	3	Specifies a value for central moment order (used for the kth central moment only)

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Output dataset

Exceptions

EXCEPTION	DESCRIPTION
Error 0017	Exception occurs if one or more specified columns have a type that is unsupported by the current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Statistical Functions](#)
[elementary](#)
[Summarize Data](#)
[A-Z Module List](#)

Compute Linear Correlation

3/10/2021 • 6 minutes to read • [Edit Online](#)

Calculates the linear correlation between column values in a dataset

Category: [Statistical Functions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Compute Linear Correlation](#) module in Azure Machine Learning Studio (classic), to compute a set of Pearson correlation coefficients for each possible pair of variables in the input dataset.

The Pearson correlation coefficient, sometimes called Pearson's R test, is a statistical value that measures the linear relationship between two variables. By examining the coefficient values, you can infer something about the strength of the relationship between the two variables, and whether they are positively correlated or negatively correlated.

How to configure Linear Correlation

Before calculating the correlation coefficient, there are some prerequisites, such as cleaning your data and verifying that the relationship between the variables is appropriate for this module. You must also remove or impute missing values.

The following restrictions apply when using this module:

- The [Compute Linear Correlation](#) module can process only numeric values. All other types of values, including missing values, non-numeric values, and categorical values, are treated as NaNs.
- Pearson's correlation is calculated for all numeric columns in the dataset that are passed as input. Be sure to exclude any columns that are inappropriate for this analysis.
- [Compute Linear Correlation](#) cannot be used with data that has missing values.

Step 1: Determine linearity

If the columns you are testing are not expected to have some kind of linear relationship, there is no point in generating this coefficient. So it is a good idea to test the columns first, to see if they have the right kind of data and the right kind of distribution in general.

There are various ways that you can determine whether the relationship between the columns is roughly linear:

- Create a scatter plot of the variables in Studio (classic), by using the **Visualize** option on the dataset. Click one of the numeric variable columns, expand **Visualizations**, and click **compare to**. Select a different variable, and a scatter plot is automatically generated. If a different type of plot is generated, it means at least one column has a different (non-numeric) data type.

- Calculate a regression equation for the two variables. There are many R packages that support this, which you can load and use in the [Execute R Script](#) module.

Step 2: Clean data

You must remove or fill in missing values, remove or clip outliers, and ensure that the columns have the proper data type.

Be sure to check for placeholders and replace such value with other appropriate values before using this module. If NaNs were inserted for missing values when the dataset was loaded from the source, it could cause an error. Placeholder values such as `999` or `-1` can also cause bad results.

To prepare your data, you can use these modules:

- [Clean Missing Data](#)
- [Clip Values](#)
- [Apply SQL Transformation](#)

You can adjust the data type of the columns by using [Edit Metadata](#). Make sure that the columns you want to analyze are marked as feature columns.

Step 3: Generate the coefficient

1. Add the [Compute Linear Correlation](#) module to your experiment. You can find this module in the [Statistical Functions](#) category in Azure Machine Learning Studio (classic).
2. Add the dataset that you want to analyze.
3. We recommend that you add a [Select Columns in Dataset](#) module between your dataset and the [Compute Linear Correlation](#) module, to remove unnecessary columns. Configure the [Select Columns in Dataset](#) module to get only the two numeric columns for which you want to compute coefficients. Otherwise, the [Compute Linear Correlation](#) module might generate many columns of NaNs.
4. There are no parameters to set for this module. However, it will fail if the columns you pass as inputs do not meet the requirements.
5. Run the experiment.

Results for two columns

Given two feature columns, the [Compute Linear Correlation](#) module returns the scalar Pearson product moment (sample) correlation coefficient. The Pearson correlation coefficient (often denoted as r) ranges in value from +1 to -1.

- `+1` indicates a strong positive linear relationship
- `-1` indicates a strong negative linear correlation
- `0` denotes no linear relationship between the two variables.

The interpretation of the coefficients depends very much on the problem you are modeling and the variables you are studying. Thus it is important to understand the context of the data when reporting and interpreting Pearson's correlation coefficient.

- If you are certain the variables are unrelated and yet the Pearson's correlation coefficient is strongly positive ($r > .5$ or so), you should investigate further.
- If you use linear correlation on two variables that you know to be perfectly correlated, and the coefficient values are not what you expect, it might indicate a problem in the data.

Results for more than two columns

Given a matrix (that is, more than two feature columns), the [Compute Linear Correlation](#) module returns a set of Pearson product moment correlations between each pair of feature columns.

Therefore, the result is an $n \times n$ table containing the coefficients for each combination of the n columns. If any columns do not meet the criteria, a NaN ("not a number" value) is returned.

For example, assume you passed in the two numeric columns `wheel-base` and `curb-weight` plus one categorical column, `make` (from the Automobile price dataset). The result is a 3x3 table of coefficients for all possible combinations of the input columns:

MAKE	WHEEL-BASE	CURB-WEIGHT
Nan	Nan	Nan
Nan	1	0.776386
Nan	0.776386	1

In this table, the rows are understood to represent each of the variables, `make`, `wheel-base`, and `curb-weight`, in that order.

- The r value for the correlation of `wheel-base` to itself is 1.
- The r value for the correlation of `wheel-base` to `curb-weight` is 0.776386.
- All correlations involving the column `make` result in NaN, including the correlation with itself, because `make` is a string feature.

We recommend that you remove non-numeric columns, to avoid complex tables with many meaningless values.

Examples

To see how this module is used in machine learning experiments, see the [Azure AI Gallery](#):

- [Data Processing and Analysis](#): This sample demonstrates multiple techniques for modifying your data. [Compute Linear Correlation](#) is used to identify potential feature columns.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

If the column that is passed as input contains scalars, the input arrays (x and y) are treated as vectors and the Pearson product moment correlation is computed as follows:

$$r = \frac{\sum_{k=1}^n (x_k - \mu_x)(y_k - \mu_y)}{\sqrt{\sum_{k=1}^n (x_k - \mu_x)^2 \sum_{k=1}^n (y_k - \mu_y)^2}}$$

In this formula, each array contains n elements and the means of the x and y samples are μ_x and μ_y respectively.

For a matrix, a matrix of data (X) is input, in which each column represents a vector of values. The data matrix should be n -by- m . The output is the m -by- m matrix, R as defined by

$$R_{i,j} = \frac{\sum_{k=1}^n (X_{k,i} - \mu_{x_i})(X_{k,j} - \mu_{x_j})}{\sqrt{\sum_{k=1}^n (X_{k,i} - \mu_{x_i})^2 \sum_{k=1}^n (X_{k,j} - \mu_{x_j})^2}}$$

In this formula, μ_x represents the mean value of the column x_i . The elements at i,j always equal 1, as they represent the correlation of a vector with itself.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Correlations matrix

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0020	Exception occurs if the number of columns in some of the datasets passed to the module is too small.
Error 0021	Exception occurs if the number of rows in some of the datasets passed to the module is too small.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Statistical Functions](#)

[A-Z Module List](#)

Evaluate Probability Function

3/10/2021 • 18 minutes to read • [Edit Online](#)

Fits a specified probability distribution function to a dataset

Category: [Statistical Functions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Evaluate Probability Function](#) module in Azure Machine Learning Studio (classic), to calculate statistical measures that describe a column's distribution, such as the Bernoulli, Pareto, or Poisson distributions.

To use this model, connect a dataset that contains at least one column of numerical values, and choose a probability distribution to test. The module returns a data table that contains values from the specified probability function.

You can compute any of these values for the chosen probability distribution:

- cumulative distribution function (**cdf**)
- inverse cumulative distribution function (**InverseCdf**)
- probability density function (**Pdf**)

Why is the probability distribution useful?

When you evaluate your data against a probability distribution, you are mapping column values against a set of values with known properties. By knowing whether your data corresponds to one of these well-known distributions, you might be able infer other properties of your data. In general, you can get better predictions from a model when you can identify the distribution that fits the data best.

The question of which probability distribution function to use depends on the data and the variables that are being measured. For example, some distributions are designed to describe probabilities of discrete values; others are intended for use only with continuous numerical variables. For some distributions, you must also know in advance an expected mean, degrees of freedom, and so forth. For details, see [Supported Probability Distributions](#)

How to configure Evaluate Probability Function

- All options change depending on the type of probability distribution you want to compute. If you change the probability distribution method, other selections you might have made are reset.

Therefore, be sure to choose the **Distribution** option first!

- The dataset used as input should contain numerical data. Other types of data are ignored.
- For each analysis, you can apply a single probability distribution method. To compute a different probability distribution, add a separate instance of the module for each distribution you intend to test.

1. Add the [Evaluate Probability Function](#) module to your experiment. You can find this module in the [Statistical Functions](#) category in Azure Machine Learning Studio (classic).
 2. Connect a dataset that contains at least one column of numbers.
 3. Use the **Distribution** option to select the kind of probability distribution that you want to calculate. See [Supported Probability Distributions](#) for a list of options and their required arguments.
 4. Set any parameters that are required by the distribution.
 5. Choose one of three statistics to create: the cumulative distribution function (**cdf**), inverse cumulative distribution function (**InverseCdf**), or Probability density function (**pdf**).
- See the [Technical notes](#) section for [definitions](#).
6. Use the column selector to choose the columns over which to compute the selected probability distribution.
 - All the columns you select must have a numerical data type.
 - The range of data in the column must also be valid, given the selected probability function. Otherwise, an error or NaN result may occur.
 - For sparse columns, any values that correspond to background zeros will not be processed.
 7. Use the **Result mode** option to specify how to output the results. You can replace column values with the probability distribution values, append the new values to the dataset, or return only the probability distribution values.
 8. Run the experiment, or right-click the [Evaluate Probability Function](#) module and click **Run selected**.

Results

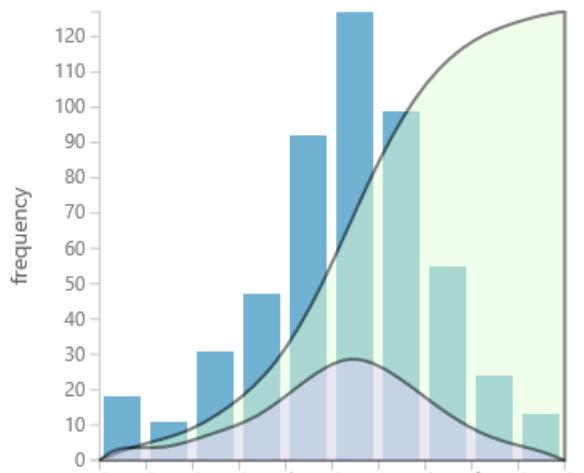
The following table contains a example of results, using the **Append** option, on a single temperature column from the **Forest Fires** sample dataset.

TEMP	STANDARDNORMAL.CDF(TEMP)	STANDARDNORMAL.PDF(TEMP)	FFISHER.CDF(TEMP)	FFISHER.CDF(TEMP)
8.2	1	1	0.984774	0.004349
18	1	1	0.997896	0.000311
14.6	1	1	0.996352	0.000648
8.3	1	1	0.985201	0.004187
11.4	1	1	0.993147	0.001502

The headings of the generated columns contain the probability distribution that was used.

If you are not sure which probability distribution is likely to suit your data, you can create a quick chart of cumulative distribution and probability density for any numeric column.

1. Right-click the dataset or module output, and select **Visualize**.
2. Select the column of interest, and in the **Histogram** pane, select **cumulative distribution** or **probability density**.
3. A chart of the distribution, like the following, is superimposed on the histogram representing the data.



- temp log scale
- frequency log scale
- bins
- cumulative distribution
- probability density

Supported probability distributions

The [Evaluate Probability Function](#) module supports the following distributions:

Bernoulli

The Bernoulli distribution is a distribution over binary values: in other words, it models the expected distribution when only two values are possible.

To calculate, select **Bernoulli**, and set the following options:

- **Probability of success**

The parameter p specifies the probability that a 1 is generated. Type a number (`float`) between 0.0 and 1.0 that specifies the probability of success. The default is .5.

Beta

The Beta distribution is a continuous univariate distribution.

To calculate, select **Beta**, and set the following options:

- **Shape**

Type a value to change the shape of the distribution.

A shape parameter is any parameter of a probability distribution that does not define its location or scale. Therefore, when you enter a value for shape, the parameter changes the shape of the distribution rather than moving, stretching, or shrinking it.

The value must be a number (`double`). The default is 1.0.

- **Scale**

Type a number to use for scaling the distribution.

By applying a scale value to the distribution, you can shrink or stretch it.

The default value is 1.0. Values must be positive numbers.

- **Upper bound**

Type a number (`double`) that represents the upper bound of the distribution. The default is 1.0.

- **Lower bound**

Type a number (`double`) that represents the lower bound of the distribution. The default is 0.0.

Binomial

The binomial distribution is a discrete univariate distribution. The binomial distribution is used to model the number of successes in a sample. Replacement is used when sampling. For sampling without replacement, use the [Hypergeometric distribution](#).

To calculate, select **Binomial**, and set the following options:

- **Probability of success**

Type a number (`float`) between 0.0 and 1.0 that indicates the probability of success. The default is .5.

- **Number of trials**

Specify the number of trials.

Use an `integer`, with a minimum value of 1. The default is 3.

Cauchy

The Cauchy distribution is a symmetric continuous probability distribution.

To calculate, select **Cauchy**, and set the following options:

- **Location**

Type a number (`double`) that represents the location of the 0th element.

By specifying a value for the **Location** parameter, you can shift the probability distribution up or down a numeric scale.

The default is 0.0.

ChiSquare

The chi-square distribution is a sum of the squares of k independent, standard, normal, random variables.

To calculate, select **ChiSquare**, and set the following options:

- **Number of degrees of freedom** Type a number (`double`) to specify the degrees of freedom. The default is 1.0.

ChiSquareRightTailed

This option provides a right-tailed chi-squared distribution.

To calculate, select **ChiSquareRightTailed**, and set the following options:

- **Number of degrees of freedom**

Type a number (`double`) to specify the degrees of freedom. The default is 1.0.

Exponential

The exponential distribution is a distribution over the real numbers parameterized by one non-negative parameter.

To calculate, select **Exponential**, and set the following options:

- **Lambda**

Type a number (`double`) to use as the lambda parameter. The default is 1.0.

FFisher

Generates the probability of the Fisher statistic for a sample, also known as the Fisher F-distribution. This distribution is two-tailed.

To calculate, select **FFisher**, and set the following options:

- **Numerator degrees of freedom**

Type a number (`double`) to specify the degrees of freedom that is used in the numerator. The default is 3.0.

- **Denominator degrees of freedom**

Type a number (`double`) to specify the degrees of freedom that is used in the denominator. The default is 6.0.

FFisherRightTailed

Creates a right-tailed Fisher distribution. The Fisher distribution is also known as the Fisher F-distribution, Snedecor distribution, or Fisher-Snedecor distribution. This particular form of the distribution is right-tailed.

To calculate, select **FFisherRightTailed**, and set the following options:

- **Numerator degrees of freedom**

Type a number (`double`) to specify the degrees of freedom that is used in the numerator. The default is 3.0.

- **Denominator degrees of freedom**

Type a number (`double`) to specify the degrees of freedom that is used in the denominator. The default is 6.0.

Gamma

The gamma distribution is a family of continuous probability distributions with two parameters. For example, chi-squared is a special case of the gamma distribution.

To calculate, select **Gamma**, and set the following options:

- **Scale**

Type a value to use for scaling the distribution.

By applying a scale value to the distribution, you can shrink or stretch it.

The default value is 1.0. Values must be positive numbers.

- **Location**

Type a number (`double`) that represents the location of the 0th element.

By specifying a value for the **Location** parameter, you can shift the probability distribution up or down a numeric scale.

The default is 0.0.

GeneralizedExtremeValues

Creates a distribution developed to handle extreme values. The generalized extreme value (GEV) distribution is actually a group of continuous probability distributions that combines the Gumbel, Fréchet, and Weibull distributions (also known as type I, II, and III extreme value distributions).

For more information about extreme value theory, see this article in Wikipedia: [Fisher-Tippet-Gnedenko theorem](#).

To calculate, select **GeneralizedExtremeValues**, and set the following options:

- **Shape**

Type a value to change the shape of the distribution.

A shape parameter is any parameter of a probability distribution that does not define its location or scale. Therefore, when you enter a value for shape, the parameter changes the shape of the distribution rather than moving, stretching, or shrinking it.

The value must be a number (`double`). The default is 1.0.

- **Scale**

Type a value to use for scaling the distribution.

By applying a scale value to the distribution, you can shrink or stretch it.

The default value is 1.0. Values must be positive numbers.

- **Location**

Type a number (`double`) that represents the location of the 0th element.

By typing a value for the **Location** parameter, you can shift the probability distribution up or down a numeric scale.

The default is 0.0.

Geometric

The geometric distribution is a distribution over positive integers parameterized by one positive real number.

To calculate, select **Geometric**, and set the following options:

- **Probability of success**

Type a number (`float`) between 0.0 and 1.0 that indicates the probability of success. The default is .5.

NOTE

This implementation of the geometric distribution does not generate zeros.

GumbelMax

The Gumbel distribution is one of several extreme value distributions. The **GumbelMax** option implements the Maximum Extreme Value Type 1 distribution.

To calculate, select **GumbelMax**, and set the following options:

- **Scale**

Type a value to use for scaling the distribution.

By applying a scale value to the distribution, you can shrink or stretch it.

The default value is 1.0. Values must be positive numbers.

- **Location**

Type a number (`double`) that represents the location of the 0th element.

By typing a value for the **Location** parameter, you can shift the probability distribution up or down a numeric scale.

The default is 0.0.

GumbelMin

The Gumbel distribution is one of several extreme value distributions. The Gumbel distribution is also referred

to as the Smallest Extreme Value (SEV) distribution or the Smallest Extreme Value (Type I) distribution. The **GumbelMin** option implements the Minimum Extreme Value Type 1 distribution.

To calculate, select **GumbelMin**, and must set the following options:

- **Scale**

Type a value to use for scaling the distribution.

By applying a scale value to the distribution, you can shrink or stretch it.

The default value is 1.0. Values must be positive numbers.

- **Location**

Type a number (`double`) that represents the location of the 0th element.

By typing a value for the **Location** parameter, you can shift the probability distribution up or down a numeric scale.

The default is 0.0.

Hypergeometric

The hypergeometric distribution is a discrete probability distribution that describes the number of successes in a sequence of n draws from a finite population without replacement, just as the binomial distribution describes the number of successes for draws with replacement.

To calculate, select **Hypergeometric**, and set the following options:

- **Number of samples**

Type an integer that indicates the number of samples to use. The default is 9.

- **Number of success**

Type an integer that defines the value for success. The default is 24.

- **Population size**

Specify the population size to use when estimating the hypergeometric distribution.

Laplace

The Laplace distribution is a distribution over the real numbers, parameterized by a mean and by a scale parameter.

To calculate, select **Laplace** distribution, and set the following options:

- **Scale**

Type a value to use for scaling the distribution.

By applying a scale value to the distribution, you can shrink or stretch it.

The default value is 1.0. Values must be positive numbers.

- **Location**

Type a number (`double`) that represents the location of the 0th element.

By typing a value for the **Location** parameter, you can shift the probability distribution up or down a numeric scale.

The default is 0.0.

Logistic

The logistic distribution is similar to the normal distribution, but it has no limit on the left side of the distribution. The logistic distribution is used in logistic regression and neural network models and for modeling life sciences data.

To calculate, select **Logistic**, and set the following options:

- **Scale**

Type a value to use for scaling the distribution.

By applying a scale value to the distribution, you can shrink or stretch it.

The default value is 1.0. Values must be positive numbers.

- **Mean**

Type a number (`double`) that indicates the estimated mean value of the distribution. The default is 0.0.

Lognormal

The lognormal distribution is a continuous univariate distribution.

To calculate, select **Lognormal**, and set the following options:

- **Mean**

Type a number (`double`) that indicates the estimated mean value of the distribution. The default is 0.0.

- **Standard deviation**

Type a positive number (`double`) that indicates the estimated standard deviation of the distribution. The default is 1.0.

NegativeBinomial

The negative binomial distribution is a distribution over the natural numbers with two parameters (`r`, `p`). In the special case that `r` is an integer, you can interpret the distribution as the number of tails before the r^{th} head when the probability of the head is `p`.

To calculate, select **NegativeBinomial**, and set the following options:

- **Probability of success**

Type a number (`float`) between 0.0 and 1.0 that indicates the probability of success. The default is .5.

- **Number of success**

Type an integer that specifies the value for success. The default is 24.

Normal

The normal distribution is also known as the Gaussian distribution.

To calculate, select **Normal**, and set the following options:

- **Mean**

Type a number (`double`) that indicates the estimated mean value of the distribution. The default is 0.0.

- **Standard deviation**

Type a positive number (`double`) that indicates the estimated standard deviation of the distribution. The default is 1.0.

Pareto

The Pareto distribution is a power-law probability distribution that coincides with social, scientific, geophysical, actuarial, and many other types of observable phenomena.

To calculate, select **Pareto**, and set the following options:

- **Shape**

Type a value (optional) to change the shape of the distribution.

A shape parameter is any parameter of a probability distribution that does not define its location or scale.

Therefore, when you enter a value for shape, the parameter changes the shape of the distribution rather than moving, stretching, or shrinking it.

The value must be a number (`double`). The default is 1.0.

- **Scale**

Type a value (optional) to change the scale of the distribution. By applying a scale value to the distribution, you can shrink or stretch it.

The value must be a number (`double`). The default is 1.0.

Poisson

In this implementation, Knuth's method is used to generate Poisson distributed random variables. For more information about the Poisson distribution, see [Poisson Regression](#).

To calculate, select **Poisson**, and set the following options:

- **Mean**

Type a number (`double`) that indicates the estimated mean value of the distribution. The default is 0.0.

Rayleigh

The Rayleigh distribution is a continuous probability distribution. As an example of how it arises, the wind speed will have a Rayleigh distribution if the components of the two-dimensional wind velocity vector are uncorrelated and normally distributed with equal variance.

To calculate, select **Rayleigh**, and set the following options:

- **Lower bound**

Type a number (`double`) that represents the lower bound of the distribution. The default is 0.0.

StandardNormal

This option provides the standard normal distribution, with no other parameters.

To calculate, select **StandardNormal**, and select the columns.

TStudent

This option implements the univariate Student's t-distribution.

To calculate, select **TStudent**, and set the following options:

- **Number of degrees of freedom**

Type a number (`double`) to specify the degrees of freedom. The default is 1.0.

TStudentRightTailed

Implements the univariate Student's t-distribution by using one right tail.

To calculate, select **TStudentRightTailed**, and set the following options:

- **Number of degrees of freedom**

Type a number (`double`) to specify the degrees of freedom. The default is 1.0.

TStudentTwoTailed

Implements a two-tailed Student's t-distribution.

To calculate, select **TStudentTwoTailed**, and set the following options:

- **Number of degrees of freedom**

Type a number (`double`) to specify the degrees of freedom. The default is 1.0.

Uniform

The uniform distribution is also known as the rectangular distribution.

To calculate, select **Uniform**, and set the following options:

- **Lower bound**

Type a number (`double`) that represents the lower limit of the distribution. The default is 0.0.

- **Upper bound**

Type a number (`double`) that represents the upper limit of the distribution. The default is 1.0.

Weibull

The Weibull distribution is widely used in reliability engineering. You can use its **Shape** parameter to model many other distributions.

To calculate, select **Weibull**, and set the following options:

- **Shape**

Type a value (optional) to change the shape of the distribution.

A shape parameter is any parameter of a probability distribution that does not define its location or scale. Therefore, when you enter a value for shape, the parameter changes the shape of the distribution rather than moving, stretching, or shrinking it.

The value must be a number (`double`). The default is 1.0.

- **Scale**

Type a value (optional) to change the scale of the distribution. By applying a scale value to the distribution, you can shrink or stretch it.

The value must be a number (`double`). The default is 1.0.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

This module supports all distributions that are provided in the open source MATH.NET Numerics library. For more information, see the documentation for the [Math.Net.Numerics.Distribution](#) library.

Right-tailed and two-tailed distributions appear as separate distributions, not as parameterized versions of base distributions. The current behavior is to preserve compatibility with Excel.

Definitions

This module supports calculating any of these values for the specified distribution:

- **cdf**, or the *cumulative distribution function*

Returns the probability for a compound event, defined as the sum of occurrences when the random variable takes a value smaller than some specific value x.

In other words, it answers the question: "How common are samples that are less than or equal to this value?"

This function can be used with both continuous and discrete numeric variables.

- **InverseCdf**, or the *inverse cumulative distribution function*

Returns the value associated with a specific cumulative probability value (cdf).

In other words, it answers the question: "What is the value of x at which the cdf function returns the

cumulative probability y ?"

- **pdf**, or the *probability density function*

Describes the relative likelihood for a random variable to be a specific value.

In other words, it answers the question: "How common are samples at exactly this value?"

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Distribution	Any	ProbabilityDistribution	StandardNormal	Select the kind of probability distribution to generate.
Method	Any	ProbabilityDistributionMethod	Cdf	Select the method to use when calculating the selected probability distribution. Options are the cumulative distribution function (cdf), the inverse cumulative distribution function (InverseCdf), and the probability density function or mass function (pdf).
Negative binomial distribution method	Any	ProbabilityDistributionMethodForNegativeBinomial	Cdf	If you select the negative binomial distribution, specify the method used for evaluating the distribution.
Probability of success	[0.0;1.0]	Float	0.5	Type a value to use as the probability of success.
Shape	Any	Float	1.0	Type a value that modifies the shape of the distribution.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Scale	$>=0.0$	Float	1.0	Type a value that changes the scale of the distribution to expand or shrink it in size.
Number of trials	$>=1$	Integer	3	Specify the number of trials.
Lower bound	Any	Float	0.0	Type a number to use as the lower limit of the distribution
Upper bound	Any	Float	1.0	Type a number to use as the upper limit of the distribution
Location	Any	Float	0.0	Type the location of the zero element in the distribution.
Number of degrees of freedom	Any	Float	1.0	Specify the number of degrees of freedom.
Numerator degrees of freedom	Any	Float	3.0	Specify the number of degrees of freedom in the numerator.
Denominator degrees of freedom	Any	Float	6.0	Specify the number of degrees of freedom in the denominator.
Lambda	$>=0.0$	Float	1.0	Specify a value for the Lambda parameter.
Number of samples	Any	Integer	9	Specify the number of samples.
Number of success	Any	Integer	24	Type a value to use as the number of success.
Population size	Any	Integer	52	Specify the population size.
Mean	Any	Float	0.0	Type the estimated mean value.
Standard deviation	$>=0.0$	Float	1.0	Type the estimated standard deviation.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Column set	Any	ColumnSelection		Choose the columns over which to calculate the probability distribution.
Result mode	Any	OutputTo	ResultOnly	Specify how the results are to be saved in the output dataset. The options are to append new columns, replace existing columns, or output only the results.

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Output dataset

Exception

For a complete list of error messages, see [Module Error Codes](#).

EXCEPTION	DESCRIPTION
Error 0017	Exception occurs if one or more specified columns have a type that is unsupported by the current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Statistical Functions](#)

[A-Z Module List](#)

Replace Discrete Values

3/10/2021 • 7 minutes to read • [Edit Online](#)

Replaces discrete values from one column with numeric values based on another column

Category: [Statistical Functions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Replace Discrete Values** module in Azure Machine Learning Studio (classic), to generate a probability score that can be used to represent a discrete value. This score can be useful for understanding the information value of the discrete values.

How it works:

You select a column that contains the discrete (or categorical) value, and then select another column to use for reference.

Depending on whether the second column is categorical or non-categorical, the module computes one of the following values:

- The **conditional probability** for the second column given the values in the first column.
- The **mean and standard deviation** for each group of values in the first column.

The module outputs both a dataset with the scores, and a function that you can save and apply to other datasets.

How to configure Replace Discrete Values

TIP

We recommend working with only one pair of columns at a time. The module does not raise an error if you select multiple columns to analyze. However, in practice, if you choose multiple columns, they are matched by an internal heuristic, not by order of selection.

Therefore, we recommend that you select a single pair of columns each time, one for **Discrete columns** and one for **Replacement columns**.

If you need to generate scores for multiple columns, use separate instances of **Replace Discrete Values**.

1. Add the **Replace Discrete Values** module to your experiment. You can find this module in the [Statistical Functions](#) group in the **experiment items** list in Azure Machine Learning Studio (classic).
2. Connect a dataset containing at least one column of categorical data.
3. **Discrete columns:** Click **Launch column selector** to choose a column that contain discrete (or categorical) values.

Any discrete columns that you select must be categorical. If you get an error, use the [Edit Metadata](#) module to change the column type.

4. **Replacement columns:** Click **Launch column selector** to choose the column that contains the values to use in computing a replacement score.

If you select multiple columns for **Discrete columns**, you must choose an equal number of replacement columns.

5. Run the experiment.

NOTE

You cannot choose which statistical function to apply. The module calculates an appropriate measure, based on the data type of the column selected for **Replacement column**.

Results

The module computes one of the following values for each pair of columns:

- If the second column contains categorical values, the module computes the **conditional probability** of the second column, given the values in the first column.

For example, assume you chose `occupation` from the **Census** dataset as the discrete column and choose `gender` as the replacement column. The output of the module would be the:

`P(gender | occupation)`

- If the second column contains non-categorical values that can be converted to numbers (such as numeric or Boolean values **not** marked as categorical), the module outputs the **mean** and **standard deviation** for each group of values in the first column.

For example, assume you use `occupation` as the **Discrete column** and the other column is the numeric column `hours-per-week`. The module would output these new values:

`Mean(hours-per-week | occupation)`

`Std-Dev(hours-per-week | occupation)`

In addition to the probability scores, the module also outputs a transformed dataset. In this dataset, the column selected as the **Replacement columns** is replaced with a column containing the computed scores.

TIP

The columns in the source dataset are not actually changed or deleted by the operation; the score columns are new ones generated by the module and output instead of the source data.

To view the source values together with the probability scores, use the [Add Columns](#) module.

Examples

The usage of **Replace Discrete Values** can be illustrated by some simple examples.

Example 1 - Replace a categorical value with a probability score

The following table contains a categorical column X, and a column Y with True/False values that are treated as categorical values. When you use **Replace Discrete Values**, it calculates a conditional probability score for the probability of Y given X, as shown in the third column.

X	Y	P(Y X)
Blue	0	$P(Y=0 X=Blue) = 0.5$
Blue	1	$P(Y=1 X=Blue) = 0.5$
Green	0	$P(Y=0 X=Green) = 2/3$
Green	0	$P(Y=0 X=Green) = 2/3$
Green	1	$P(Y=1 X=Green) = 1/3$
Red	0	$P(Y=0 X=Red) = .75$
Red	0	$P(Y=0 X=Red) = .75$
Red	1	$P(Y=1 X=Red) = .25$
Red	0	$P(Y=0 X=Red) = .75$

Example 2 - Calculate mean and standard deviation based on a noncategorical column

When the second column is numerical, Replace Discrete Values calculates the mean and standard deviation instead of a conditional probability score.

The following example is based on the **Auto Prices** sample dataset, simplified as follows:

- A small subset of columns was selected.
- Only the top 30 rows were extracted, by using the **Head** option of the **Partition and Sample** module.
- The **Replace Discrete Values** module was used to compute the **mean** and **standard deviation** for vehicle curb weight, given the categorical column, `num-of-doors`.

The following table illustrates the results:

BODY	NUM-OF-DOORS	CURB-WEIGHT	MEAN(CURB-WEIGHT NUM-OF-DOORS)	STD-DEV(CURB-WEIGHT NUM-OF-DOORS)
std	two	2548	2429.785714	507.45699
std	four	2337	2625.6	493.409877
std	two	2507	2429.785714	507.45699
turbo	four	3086	2625.6 5	493.409877
std	four	1989	2625.6	493.409877
turbo		2191		
std	four	2535	2625.6	493.409877

You can verify the mean for each group of values by using the `AVERAGEIF` function in Excel.

Example 3 - Handling missing values

This example demonstrates how missing values (nulls) propagate to the results when conditional probability scores are calculated.

- If the discrete value column and the calculation lookup column contains any missing values, the missing values are propagated to the new column.
- If the discrete value column contains only missing values, the module cannot process the column and an error message appears.

X	Y	P(Y X)
1	True	P(Y=true X=1) = 1/2
1	False	P(Y=false X=1) = 1/2
2	True	P(Y=true X=2) = 1/3
2	False	P(Y=false X=2) = 1/3
2	Null	P(Y=null X=2) = null

Technical notes

- You must ensure that any discrete columns you want to replace are categorical, or the module will return an error. To do this, use the [Edit Metadata](#) module.
- If the second column contains Boolean values, the True-False values are processed as numeric with FALSE and TRUE equivalent to 0 and 1 respectively.
- The formula for the standard deviation column calculates the population standard deviation. Therefore, N is used in the denominator instead of (N - 1).
- If the second column contains noncategorical data (numeric or Boolean values), the module computes the mean and standard deviation of Y for the given value of X.

That is, for each row in the dataset indexed by i :

$$\text{Mean}(Y|X)_i = \text{Mean}(Y|X = X_i)$$

$$\text{StdDev}(Y|X)_i = \text{StdDev}(Y|X = X_i)$$

- If the second column contains categorical data or values that are neither numeric nor Boolean, the module computes the conditional probability of Y for the given value of X.
- Any Boolean values in the second column are processed as numeric data with FALSE and TRUE equivalent to 0 and 1 respectively.
- If there is a class in the discrete column, such that a row with a missing value is present in the second column, the sum of conditional probabilities within the class is less than one.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

NAME	TYPE	DESCRIPTION
------	------	-------------

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Discrete columns	Any	ColumnSelection		Selects the columns that contain discrete values
Replacement columns	Any	ColumnSelection		Selects the columns that contain the data to use in place of the discrete values

Outputs

NAME	TYPE	DESCRIPTION
Supplemented dataset	Data Table	Dataset with replaced data
Transform function	ITransform interface	Definition of the transform function, which can be applied to other datasets

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of the data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0020	Exception occurs if the number of columns in some of the datasets passed to the module is too small.
Error 0021	Exception occurs if the number of rows in some of the datasets passed to the module is too small.
Error 0017	Exception occurs if one or more specified columns have a type that is unsupported by the current module.
Error 0026	Exception occurs when columns with the same name are not allowed.
Error 0022	Exception occurs if the number of selected columns in the input dataset does not equal the expected number.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Statistical Functions](#)

Summarize Data

3/10/2021 • 4 minutes to read • [Edit Online](#)

Generates a basic descriptive statistics report for the columns in a dataset

Category: [Statistical Functions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Summarize Data** module in Azure Machine Learning Studio (classic), to create a set of standard statistical measures that describe each column in the input table.

Such summary statistics are useful when you want to understand the characteristics of the complete dataset. For example, you might need to know:

- How many missing values are there in each column?
- How many unique values are there in a feature column?
- What is the mean and standard deviation for each column?

The module calculates the important scores for each column, and returns a row of summary statistics for each variable (data column) provided as input.

TIP

You might already know that you can get a short list of statistics by using the **Visualize** option in Studio (classic). However, this visualization is created based on some top number of rows. In contrast, the **Summarize Data** module computes its statistics on all rows of data.

How to use Summarize Data

1. Add the **Summarize Data** module to your experiment. You can find this module in the [Statistical Functions](#) category in Studio (classic).
2. Connect the dataset for which you want to generate a report.
If you want to report on only some columns, use the **Select Columns in Dataset** module to project a subset of columns to work with.
3. No additional parameters are required. By default, the module analyzes all columns that are provided as input, and depending on the type of values in the columns, outputs a relevant set of statistics as described in the **Results** section.
4. Run the experiment, or right-click the module, and select **Run selected**.

Results

The report from the module can include the following statistics.

- The exact statistics that are generated depend on the column data type. See the [Technical notes](#) section for details.
- The assumption is made that the instances belong to a representative sample of a population. If you need to compute statistics on a population, use the options in the [Compute Elementary Statistics](#) module, which can compute either sample or population statistics.

COLUMN NAME	DESCRIPTION
Feature	Name of the column
Count	Count of all rows
Unique Value Count	Number of unique values in column
Missing Value Count	Number of unique values in column
Min	Lowest value in column
Max	Highest value in column
Mean	Mean of all column values
Mean Deviation	Mean deviation of column values
1st Quartile	Value at first quartile
Median	Median column value
3rd Quartile	Value at third quartile
Mode	Mode of column values
Range	Integer representing the number of values between the maximum and minimum values
Sample Variance	Variance for column; see Note
Sample Standard Deviation	Standard deviation for column; see Note
Sample Skewness	Skewness for column; see Note
Sample Kurtosis	Kurtosis for column; see Note
P0.5	0.5% percentile
P1	1% percentile
P5	5% percentile
P95	95% percentile

COLUMN NAME	DESCRIPTION
P99.5	99.5% percentile

TIP

Output the statistics report as a tabular dataset, so that you can use the data in BI reporting tools, or use the values as input to another operation in the experiment.

Examples

For examples of how to use the **Summarize Data** module in an experiment, see the [Azure AI Gallery](#):

- [Download dataset from UCI](#): Reads a dataset in CSV format by using its URL in the UCI Machine Learning Repository, and generates some basic statistics about the dataset.
- [Dataset Processing and Analysis](#): Loads the dataset into the workspace, changes column names, and adds metadata.
- [Prediction of student performance](#): Reads data stored in TSV format from Azure Blob storage.

Technical notes

- For numeric and Boolean columns, you can output the mean, median, mode, and standard deviation.
- For non-numeric columns, only the values for **Count**, **Unique value count**, and **Missing value count** are computed. For other statistics, a null value is returned.
- Columns that contain Boolean values are processed using these rules:
 - When calculating **Min**, a logical AND is applied.
 - When calculating **Max**, a logical OR is applied
 - When computing **Range**, the module first checks whether the number of unique values in the column equals 2.
 - When computing any statistic that requires floating-point calculations, values of True are treated as 1.0, and values of False are treated as 0.0.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Output

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	A profile of the input dataset that contains descriptive statistics

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more inputs are null or empty.
Error 0020	Exception occurs if the number of columns in some of the datasets passed to the module is too small.
Error 0021	Exception occurs if the number of rows in some of the datasets passed to the module is too small.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Statistical Functions](#)

[Compute Elementary Statistics](#)

Test Hypothesis Using t-Test

3/10/2021 • 8 minutes to read • [Edit Online](#)

Compares means from two columns using a t-test

Category: [Statistical Functions](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the [Test Hypothesis Using t-Test](#) module in Azure Machine Learning Studio (classic), to generate scores for three types of t-tests:

- Single sample t-test
- Paired t-test
- Unpaired t-test

In general, a t-test helps you compare whether two groups have different means. For example, suppose you are evaluating trial data for patients who received Drug A vs. patients who received Drug B, and you need to compare a recovery rate metric for both groups. The *null hypothesis* would assume that the recovery rate is the same in both groups, and furthermore, that the values for the recovery rate have a normal distribution in both two groups.

By using [Test Hypothesis Using t-Test](#) and providing the columns that contain the recovery rates as input, you can get scores that indicate whether the difference is meaningful, which would signify that the null hypothesis should be rejected. The test takes into account factors such as how big the difference is between the values, the size of the sample (larger is better), and how big the standard deviation is (lower is better).

By reviewing the results of the [Test Hypothesis Using t-Test](#) module, you can determine whether the null hypothesis is TRUE or FALSE, and review the confidence (P) scores from the t-test.

How to choose a t-test

Choose a **single sample t-test** when these conditions apply:

- You have a single sample of scores.
- All scores are independent from each other.
- The sampling distribution of \bar{x} is normal.

In general, the single sample t-test is used to compare an average value to a known number.

Choose a **paired t-test** when these conditions apply:

- You have a matched pairs of scores. For example, you might have two different measures per person, or matched pairs of individuals (such as a husband and wife).
- Each pair of scores is independent of every other pair.

- The sampling distribution of d is normal.

A paired t-test is useful when comparing related cases. By averaging the differences between the scores of the paired cases, you can determine whether the total difference is statistically significant.

Choose an **unpaired t-test** when these conditions apply:

- You have two independent samples of scores. That is, there is no basis for pairing scores in sample 1 with those in sample 2.
- All scores within a sample are independent of all other scores within that sample.
- The sampling distribution of $x_1 - x_2$ is normal.
- Optionally, satisfy the requirement that the variance among the groups be roughly equal.

How to configure Test Hypothesis Using t-Test

Use a single dataset as input. The columns that you are comparing must be in the same dataset.

If you need to compare columns from different datasets, you can isolate each column to compare by using [Select Columns in Dataset](#), and then merge them into one dataset by using [Add Columns](#).

1. Add the [Test Hypothesis Using t-Test](#) module to your experiment.

You can find this module in the [Statistical Functions](#) category in Studio (classic).

2. Add the dataset that contains the column or columns that you want to analyze.

3. Decide which kind of t-test is appropriate for your data. See [How to choose a t-test](#).

4. **Single sample:** If you are using a single sample, set these parameters:

- **Null hypothesized μ :** Type the value to use as the null-hypothesized mean for the sample. This specifies the expected mean value against which the sample mean will be tested.
- **Target column:** Use the Column Selector to choose a single numeric column for testing.
- **Hypothesis type:** Choose a one-tail or two-tail test. The default is a two-tailed test. This is the most common type of test, in which the expected distribution is symmetric around zero.

The **One Tail GT** option is for a one-tailed greater than test. This test gives more power to detect an effect in one direction, by not testing the effect in the other direction.

The **One Tail LT** option gives a one-tailed less than test.

- **a:** Specify a confidence factor. This value is used to evaluate the value of P (the first output of the module). If p is lower than the confidence factor, the null hypothesis is rejected.

5. **PairedSamples:** If you are comparing two samples from the same population, set these parameters:

- **Null hypothesized μ :** Type a value that represents the sample difference between the pair of samples.
- **Target column:** Use the Column Selector to choose the two numeric columns to test.
- **Hypothesis type:** Select either a one-tail or two-tail test. The default is a two-tailed test.
- **a:** Specify the confidence factor. This value is used to evaluate the value of P (the first output of the module). If p is lower than the confidence factor, the null hypothesis is rejected.

6. **UnpairedSamples:** If you compare two unpaired samples, set these parameters:

- **Assume equal variance:** Deselect this option when the samples are from different populations.
- **Null hypothesized μ_1 :** Type the mean for the first column.
- **Null hypothesized μ_2 :** Type the mean for the second column.
- **Target columns:** Use the Column Selector to choose two numeric columns to test.
- **Hypothesis type:** Indicate whether the test is one-tail or two-tail. The default is a two-tailed test.
- **α :** Specify the confidence factor. This value is used to evaluate the value of P (the first output of the module). If p is lower than the confidence factor, the null hypothesis is rejected.

7. Run the experiment.

Results

The output of the module is a dataset containing the t-test scores, and a transformation that you can optionally save to re-apply to this or another dataset using [Apply Transformation](#).

The dataset of scores contains these values, regardless of the type of t-test you used:

- A probability score that indicates the confidence of the null hypothesis
- A value that indicates whether the Null hypothesis should be rejected

TIP

Remember, the goal is to determine whether you can reject the null hypothesis. A score of 0 doesn't mean you should accept the null hypothesis: it means that you don't have enough data, and need further investigation.

Technical notes

The module automatically names the output columns according to the following conventions, depending on which type of t-test was selected, and whether the result was to reject or accept the null hypothesis.

Given input columns with names {0} and {1}, the module creates the following names:

COLUMNS	SINGLESAMPLESET	PAIREDSAMPLES	UNPAIREDSAMPLES
Output column P	P_ss({0})	P_ps({0}, {1})	P_us({0}, {1})
Output column RejectH0	RejectH0_ss({0})"	RejectH0_ps({0}, {1})	RejectH0_us({0}, {1})

How Scores are computed

This module computes and uses the sample standard deviation; therefore, the equation uses $(n-1)$ in the denominator.

Computing scores for a single-sample test

Given a single sample of scores, all independent of each other, and a normal distribution, the score is calculated as follows:

1. Take the following input:
 - A single column of values from the dataset
 - The null hypothesis (H_0) parameter μ_0
 - The confidence score specified by α
2. Extract the number of samples (n).

3. Calculate the mean of the sample data.
4. Calculate the standard deviation (s) of the sample data.
5. Calculate t and degrees of freedom (df):

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}, \quad df = n - 1$$

6. Extract probability P from distribution table T by using t and df .

Computing scores for a paired t-test

Given a matched set of scores, with each pair independent of the other, and a normal distribution in each set, the score is calculated as follows:

1. Take the following input:
 - Two columns of values from the dataset
 - The null hypothesis (H_0) parameter d_0
 - The confidence score specified by α
2. Extract some number of sample pairs (n).
3. Calculate the mean of differences for the sample data:

$$(\bar{d} = \overline{x_1 - x_2})$$

4. Calculate the standard deviation of differences (sd).
5. Calculate t and the degrees of freedom (df):

$$t = \frac{\bar{d} - d_0}{s_d / \sqrt{n}}, \quad df = n - 1$$

6. Extract probability (P) from the distribution table (T) by using t and df .

Computing scores for an unpaired t-test

Given two independent samples of scores, with a normal distribution of values in each sample, the score is calculated as follows:

1. Take the following input:
 - A dataset that contains two columns of `doubles`
 - The null hypothesis (H_0) parameter (d_0)
 - The confidence score specified by α
2. Extract a number of samples in each group, n_1 and n_2 .
3. Calculate the means for each of the sample sets.
4. Calculate the standard deviation for each group as s_1 and s_2 .
5. Calculate t and degrees of freedom (df):

Optionally, satisfy the requirement that the variance among the groups be roughly equal, as follows:

1. Calculate the pooled standard deviation first:

$$S_p^2 = \frac{(n_1 - 1) S_1^2 + (n_2 - 1) S_2^2}{n_1 + n_2 - 2}$$

2. If there is no assumption about variance equality, calculate as follows:

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - d_0}{S_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}, \quad df = n_1 + n_2 - 2$$

3. Extract P from the distribution table (T) by using t and df.

Computing the null hypothesis

The probability of the null hypothesis, designated as P, is calculated as follows:

- If $P < \alpha$, set the Reject flag to True.
- If $P \geq \alpha$, set the Reject flag to False.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Hypothesis type	Any	Hypothesis	Two-tail	Student's t-test null hypothesis type
Null hypothesized μ	Any	Float	0.0	For the single sample t-test, the null-hypothesized mean for the sample For the paired t-test, the sample difference
Target column(s)	Any	ColumnSelection	None	Target column(s) selection pattern
Assume equal variances	Any	Boolean	True	Assume variances of two samples are equal Applies only to unpaired samples
Null hypothesized μ_1	Any	Float	0.0	Null-hypothesized mean for first sample

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
a	[0.0;1.0]	Float	0.95	Confidence factor (if P is lower than the confidence factor, null hypothesis is rejected)

Outputs

NAME	TYPE	DESCRIPTION
P	Data Table	A probability score that indicates the confidence of the null hypothesis
Reject H0	Data Table	Value that indicates whether the Null hypothesis should be rejected

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0008	Exception occurs if parameter is not in range.
Error 0017	Exception occurs if one or more specified columns have a type that is unsupported by the current module.
Error 0020	Exception occurs if the number of columns in some of the datasets passed to the module is too small.
Error 0021	Exception occurs if the number of rows in some of the datasets passed to the module is too small.
Error 0031	Exception occurs if the number of columns in column set is less than needed.
Error 0032	Exception occurs if the argument is not a number.
Error 0033	Exception occurs if the argument is infinity.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Statistical Functions](#)

Text Analytics

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the text analytics modules included in Azure Machine Learning Studio (classic). These modules provide specialized computational tools for working with both structured and unstructured text, including:

- Multiple options for preprocessing text.
- Language detection.
- Creation of features from text using customizable n-gram dictionaries.
- Feature hashing, to efficiently analyze text without preprocessing or advanced linguistic analysis.
- Vowpal Wabbit, for very fast machine learning on text. Vowpal Wabbit supports feature hashing, topic modeling (LDA), and classification.
- Named entity recognition, to extract the names of people, places, and organizations from unstructured text.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Examples

For examples of text analytics using Azure Machine Learning, see the [Azure AI Gallery](#):

- [News categorization](#): Uses feature hashing to classify articles into a predefined list of categories.
- [Find similar companies](#): Uses the text of Wikipedia articles to categorize companies.
- [Text classification](#): Demonstrates the end-to-end process of using text from Twitter messages in sentiment analysis (five-part sample).

List of modules

The **Text Analytics** category in Azure Machine Learning Studio (classic) includes these modules:

- [Detect Languages](#): Detects the language of each line in the input file.
- [Extract Key Phrases from Text](#): Extracts key phrases from given text.
- [Extract N-Gram Features from Text](#): Creates N-Gram dictionary features, and does feature selection on them.
- [Feature Hashing](#): Converts text data to integer-encoded features by using the Vowpal Wabbit library.
- [Latent Dirichlet Allocation](#): Performs topic modeling by using the Vowpal Wabbit library for LDA.
- [Named Entity Recognition](#): Recognizes named entities in a text column.
- [Preprocess Text](#): Performs cleaning operations on text.
- [Score Vowpal Wabbit 7-4 Model](#): Scores input from Azure by using version 7-4 of the Vowpal Wabbit machine learning system.
- [Score Vowpal Wabbit 7-10 Model](#): Scores input from Azure by using version 7-10 of the Vowpal Wabbit machine learning system.
- [Score Vowpal Wabbit 8 Model](#): Scores input from Azure by using version 8 of the Vowpal Wabbit machine learning system.

learning system.

- [Train Vowpal Wabbit 7-4 Model](#): Trains a model by using version 7-4 of the Vowpal Wabbit machine learning system.
- [Train Vowpal Wabbit 7-10 Model](#): Trains a model by using version 7-10 of the Vowpal Wabbit machine learning system.
- [Train Vowpal Wabbit 8 Model](#): Trains a model by using version 8 of the Vowpal Wabbit machine learning system.

See also

- [Regression module](#)
- [Classification module](#)
- [Clustering module](#)
- [OpenCV Library Modules category](#)
- [All module categories](#)

Detect Languages

3/10/2021 • 4 minutes to read • [Edit Online](#)

Detects the language of each line in the input file

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module Overview

This article describes how to use the **Detect Languages** module in Azure Machine Learning Studio (classic) to analyze text input and identify the language associated with each record in the input.

The language detection algorithm can identify many different languages. Just specify the string column to analyze, and the total number of languages to detect. The algorithm will analyze each row of text, and assign a probability score for each language. The language in the first result column is the language that got the highest score.

How to configure Detect Languages

1. Add the dataset containing the text you want to analyze to an experiment in Azure Machine Learning Studio (classic). The column with the text to analyze must be the string data type.

The dataset need not contain a label column; the language detection algorithm works purely on linguistic features of the supported languages.

If you are importing new data, make sure that your data is saved in the UTF-8 format. Other Unicode formats are not supported.

2. Add the **Detect Languages** module to your experiment, and connect the dataset with the text for language detection.
3. For **Text column**, choose the column you want to analyze.
4. For **Upper bound on number of languages to detect**, indicate the maximum number of languages to detect.
Setting an upper bound on the number of languages can improve performance.

5. Run the experiment.

Results

The **Detect Languages** module outputs a language identifier and score for each row.

For example, the following table contains a sample analysis on test data.

- The first two columns *col1* and *language label* are columns passed through from the input dataset. In this example, because the input dataset was designed for testing the module, the expected language was

already known, and is provided in the label column.

- The remaining columns are generated by the **Detect Languages** module. If there are equi-probable language matches, several languages might be listed, with a score for each. In this case, the module predicts just one language for each row, together with the probability score for that language.

If the module fails to detect any language with a sufficiently high score, a result of (Unknown) with a score of 0 is output. However, the languages supported by the module can change over time as the API is updated.

COL1	LANGUAGE LABEL	COL1 LANGUAGE	COL1 ISO6391 LANGUAGE	COL1 ISO6391 LANGUAGE SCORE
It was a wonderful hotel with a friendly staff and good service	English	English	en	100
Es war ein wunderbares Hotel mit freundlichem Personal und guter service	German	German	de	100
C'est un magnifique hôtel avec un personnel sympathique et un service de qualité	French	French	fr	100
Det var et dejligt hotel med et venligt personale og god service	Danish	Danish	nl	100
Va ser un magnífic hotel amb un personal amable i bon servei	Catalan	Catalan	ca	92.30769348
とても素敵なおホテルで、スタッフは親切で、サービスもよかったです	Japanese	(Unknown)		0
qu mebpa'mey naQ friendly QaQ chavmoH je	Klingon	French	fr	77.5

Examples

For examples of how the **Detect Languages** module is used in an experiment, see the [Azure AI Gallery](#):

- [Filter Movie Titles by Language](#): Detects the language used in movie names, and then uses the language identifier to split the dataset into English vs non-English movies.

Technical notes

For a general idea of the languages that potentially can be detected, refer to [Bing Translator](#).

Many more languages can be detected than Azure Machine Learning currently supports for advanced text analytics. We recommend that you use the results of **Detect Languages** to filter the results that you send to other modules that require language-specific processing.

The underlying linguistic services are also used by the [Text Analytics](#) service in [Azure Cognitive Services](#).

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	The input

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Upper bound on number of languages to detect	Integer	[1;184]	Required	1	Upper bound on number of languages to detect.
Text column	ColumnSelection		Required		Name or one-based index of text column.

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	The result

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0010	Exception occurs if input datasets have column names that should match but do not.
Error 0016	Exception occurs if input datasets passed to the module should have compatible column types but do not.
Error 0008	Exception occurs if parameter is not in range.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Text Analytics](#)

[A-Z Module List](#)

Extract Key Phrases from Text

3/10/2021 • 4 minutes to read • [Edit Online](#)

Extracts key phrases from given text

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article explains how to use the **Extract Key Phrases from Text** module in Azure Machine Learning Studio (classic), to pre-process a text column. Given a column of natural language text, the module extracts one or more meaningful phrases. A phrase might be a single word, a compound noun, or a modifier plus a noun.

This module is a wrapper for natural language processing APIs for key-phrase extraction. The phrases are analyzed as potentially meaningful in the context of the sentence for various reasons:

- The phrase captures the topic of the sentence.
- The phrase contains a combination of modifier and noun that indicates sentiment.

For example, assume the sentence analyzed is: "It was a wonderful hotel to stay at, with unique decor and friendly staff."

The **Extract Key Phrases from Text** module might return these key phrases:

- wonderful hotel
- friendly staff
- unique decor

How to configure Extract Key Phrases from Text

To extract key phrases, you must connect a dataset that has a column of text.

1. Add the **Extract Key Phrases from Text** module to your experiment in Azure Machine Learning Studio (classic). Then, connect a dataset that has at least one full-text column.
2. Use the Column Selector to select a column of type string, from which to extract key phrases.
3. For **Language**, select a language to use when analyzing phrases. If you specify a language, only phrases in the target language will be output.
4. If the text column contains phrases in multiple languages, choose the option, **Language identified in columns**. A new column selector is displayed that lets you select a column in your data set that contains a language identifier. The language identifier can either be the language name or the Iso6391 culture identifier. For example, either "English" or "en" are acceptable.

TIP

Before running **Extract Key Phrases from Text**, use the [Detect Languages](#) module to identify the language in each row and generate the identifier for you. An error is raised if the language identifier column contains any languages not supported by **Extract Key Phrases from Text**.

Results

The output of the module is a dataset containing a column of comma-separated key phrases.

For example, the following example results are for an input dataset containing reviews in multiple languages:

KEY PHRASES

novel,nuclear submarine,good book,adventure story,avalanche of events,good characters

primer misterio,personajes,fan,aventura,isla

- All output phrases are contained in a single column; no other columns are passed through, and an identifier is not added. However, if you want to align the output phrases with the source text, you can recombine the output phrases with the input by using the [Add Columns](#) module.
- The output of key-phrase extraction does not flag the language of individual phrases.
- If a language is included that is not supported by the **Extract Key Phrases** module, an error is raised (0039). To avoid errors, be sure to filter out input text that has an incompatible language identifier.

If there are very few rows of other languages, you can also avoid the error by omitting the language identifier, and analyzing all text using a single language selection. However, when you do so, results are very poor, because entire sentences in the other languages might be output as a single key phrase.

Examples

The following example demonstrates how to use this module to extract key phrases and then build a word cloud from the phrases: [Extract Key Phrases and Show Word Cloud](#)

See the [Azure AI Gallery](#) for more examples of text processing using Azure Machine Learning.

Technical notes

This module currently supports the following languages:

- Dutch
- English
- French
- German
- Italian
- Spanish

For additional languages, consider using the [Text Analytics API](#) in Azure Cognitive Services. For more information, see [How to extract key phrases in Text Analytics](#)

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	The table containing the text to be processed.

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Culture-language column	ColumnSelection		language:Column contains language		Name or one-based index of the column containing the culture-language information
Text column	ColumnSelection		Required		Name or one-based index of the text column.
Language	T_Language	English, Spanish, French, Dutch, German, Italian, Column contains language	Required	English	Select the language of the text to be processed.

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	The extracted key phrases

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0010	Exception occurs if input datasets have column names that should match but do not.
Error 0016	Exception occurs if input datasets passed to the module should have compatible column types but do not.
Error 0008	Exception occurs if parameter is not in range.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Text Analytics](#)

[A-Z Module List](#)

Extract N-Gram Features from Text

3/10/2021 • 15 minutes to read • [Edit Online](#)

Creates N-Gram dictionary features and does feature selection on them

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article explains how to use the **Extract N-Gram Features from Text** module in Azure Machine Learning Studio (classic), to *featurize* text, and extract only the most important pieces of information from long text strings.

The module works by creating a dictionary of n-grams from a column of free text that you specify as input. The module applies various information metrics to the n-gram list to reduce data dimensionality and identify the n-grams that have the most information value.

If you have already created a vocabulary of n-grams, you can update its statistics, or merge in new terms, using a weighting algorithm of your choice.

Because this module supports featurization from n-grams, it can also be used when scoring.

How to configure Extract N-Gram Features from Text

Those module supports the following scenarios for creating, updating, or applying an n-gram dictionary:

- You are developing a new model using a column of free text column and want to extract text features based purely on the input data. [See instructions](#).
- You have an existing set of text features, and want to update the weights by processing new text inputs. [See instructions](#).
- You are generating scores from a predictive model and need to generate and use text inputs with an n-gram dictionary as part of the scoring process. [See instructions](#).

You could use the [example experiment](#) for reference.

Create a new n-gram dictionary from a text column

1. Add the **Extract N-Gram Features from Text** module to your experiment and connect the dataset that has the text you want to process.
2. For **Text column**, choose a column of type **string** that contains the text you want to extract.

By default, the module selects all string columns. However, because the result are verbose, you might need to process a single column at a time.

3. For **Vocabulary mode**, select **Create** to indicate that you are creating a new list of n-gram features.

For information about how to update an existing set of n-gram features, see [this section](#).

4. For **N-Grams size**, type a number that indicates the *maximum* size of the n-grams to extract and store.

For example, if you type , unigrams, bigrams, and trigrams will be created.

5. For **K-Skip size**, type the maximum number of characters that can be different when identifying variants of n-grams. If the value of k is set to 0, n-grams can be created only from a unique, contiguous sequence of characters.

For example, assume that your dictionary contains the unigram "computer". A k value of 0 would mean that "computer" is the only valid unigram. If you increase the value of k to 1, you can skip over one intervening character, which lets you find more similar sequences. A skip-gram with a k value of 1 would differ by one character from the 0- k unigram. Thus, the skip-grams "computer" and "compuuter" would both be considered part of the same dictionary entry as "computer". Setting the k value to 2 would match even more dissimilar words.

For more information about how skip-grams are used in text analytics, see this paper: [Candidate Generation and Feature Engineering for Supervised Lexical Normalization](#)

6. The option, **Weighting function**, is required only if you merge or update vocabularies. It specifies how terms in the two vocabularies and their scores should be weighted against each other.

7. For **Minimum word length**, type the minimum word length of strings that can be analyzed.

For example, assume the minimum word length was set to 3 (the default value), and you had one input that had a single word, and another that had some short text like "nice place". Both rows would be ignored.

8. For **Maximum word length**, type the maximum number of letters that can be used in any *single word* in an n-gram.

By default, up to 25 characters per word or token are allowed. Words longer than that are removed, on the assumption that they are possibly sequences of arbitrary characters rather than actual lexical items.

9. For **Minimum n-gram document absolute frequency**, type a number that indicates the minimum occurrences required for any single word or token to be included in the n-gram dictionary.

For example, if you use the default value of 5, any n-gram or skip-gram must appear at least five times in the corpus to be included in the n-gram dictionary.

10. For **Maximum n-gram document ratio**, type a number that represents this ratio: the number of rows that contain a particular n-gram, over the number of rows in the overall corpus.

For example, a ratio of 1 would indicate that, even if a specific n-gram is present in every row, the n-gram can be added to the n-gram dictionary. More typically, a word that occurs in every row would be considered a noise word and would be removed. To filter out domain-dependent noise words, try reducing this ratio.

IMPORTANT

The rate of occurrence of particular words is not uniform, but varies from document to document. For example, if you are analyzing customer comments about a specific product, the product name might be very high frequency and close to a noise word, but be a significant term in other contexts.

11. Select the option, **Detect out-of-vocabulary rows**, if you want to generate an indicator for any rows that contain words not in the n-gram vocabulary, which are called "out of vocabulary" (OOV) words.

All lexicons are finite; therefore, your text corpus is almost guaranteed to include words that are not in the lexicon or n-gram dictionary. However, such words can have various effects on language models, including higher error rates compared to in-vocabulary (IV) words. Depending on your domain, these OOV words might represent important content words.

By identifying rows that contain these words, you can either compensate for the effects of these terms, or handle the terms and related rows separately.

12. Select the option, **Mark begin-of-sentence**, to add a special character sequence that indicates the beginning of a sentence in your n-gram dictionary. Prefixing n-grams that start a sentence with a special character is common in text analysis and can be useful in analyzing discourse boundaries.

Azure ML Studio (classic) inserts the symbol . You cannot specify a custom character.

13. Select the option **Normalize n-gram feature vectors** if you want to normalize the feature vectors. When you do this, each n-gram feature vector is divided by its L2 norm.

Normalization is used by default.

14. Set **Use filter-based feature selection** to **True** if you want to enable additional options for managing the size of your text feature vector.

- Feature selection can be helpful in reducing the dimensionality of your n-grams.
- When you do not apply filter selection, all possible n-grams are created, increasing coverage at the expense of making the dictionary longer and possibly including many infrequent terms.
- In a small corpus, using feature selection can greatly reduce the number of terms that are created.
- For more information, see [Filter Based Feature Selection](#).

If you are using feature selection, you must select a method from the **Feature scoring method** dropdown list:

- **PearsonCorrelation**: Computes Pearson's correlation based on the label column value and the text vector.
- **MutualInformation**: Computes a mutual information score, based on the label column value and the text vector.
- **KendallCorrelation**: Computes Kendall's correlation, based on the label column value and the text vector.
- **SpearmanCorrelation**: Computes the Spearman correlation, based on the label column value and the text vector.
- **ChiSquared**: Uses the chi-squared method to calculate the correlation between the label column value and the text vector.
- **FisherScore**: Computes the Fisher score for the label column value and the text vector.
- **Count-based feature selection**: Creates new features based on the counts of values. A label column is not required with this method.

Depending on the method you choose, set one of the following options:

- **Number of desired features**: Required if you use any feature selection method other than count-based feature selection.

In the process of feature selection, all n-grams get a feature score, and n-grams are ranked by score. The value you set here determines how many of the most highly-ranked features are output. N-grams with lower feature scores are discarded.

- **Minimum number of non-zero elements**: Required if you use count-based feature selection.

Type a whole number that represents the minimum number of total instances required to tabulate counts for a potential feature.

15. Run the experiment.

See [this section](#) for an explanation of the results and their format.

Update an existing n-gram dictionary, or merge dictionaries

1. Add the **Extract N-Gram Features from Text** module to your experiment and connect the dataset that has the text you want to process to the **Dataset** port.
2. For **Text column**, choose the text column that contains the text you want to featurize. By default, the module selects all columns of type string. For best results, process a single column at a time.
3. Add the saved dataset containing a previously generated n-gram dictionary, and connect it to the **Input vocabulary** port. You can also connect the **Result vocabulary** output of an upstream instance of the **Extract N-Gram Features from Text** module.

To merge or update vocabulary, the schema of the input vocabulary must exactly match the expected format. Do not remove any columns from or add any columns to the input vocabulary.

4. For **Vocabulary mode**, select one of the following update options from the drop-down list:

- **ReadOnly**: Represents the input corpus in terms of the input vocabulary. That is to say, rather than computing term frequencies from the new text dataset (on the left input), the n-gram weights from the input vocabulary are applied as is.

TIP

Use this option when scoring a text classifier.

- **Update**: Creates a new n-gram vocabulary from the input corpus, and merges it with the input vocabulary. In other words, you can add new entries to the created vocabulary from the input vocabulary, or you can update existing entries.

TIP

Use this option for incremental updates of vocabulary with incoming data batches.

- **Merge**: Generates a new n-gram vocabulary from the input corpus.

This option is useful if you are passing a background vocabulary as input to the module and want to reduce the weight of stop words. In other words, each entry that has a high document frequency score in the background vocabulary will be assigned a lower inverse document frequency score in the created vocabulary.

TIP

Use this option if you don't want to add new entries to the created vocabulary from the input, and only want to adjust the scores of existing entries.

5. The option, **Choose the weighting function**, is required if you merge or update vocabularies. The weighting function specifies how the DF and IDF scores in the two vocabularies should be weighted against each other:

- **Binary Weight**: Assigns a binary presence value to the extracted n-grams. In other words, the value for each n-gram is 1 when it exists in the given document, and 0 otherwise.
- **TF Weight**: Assigns a term-frequency score (TF) to the extracted n-grams. The value for each n-gram

is its occurrence frequency in the given document.

- **IDF Weight:** Assigns an inverse document frequency score (**IDF**) to the extracted n-grams. The value for each n-gram is the log of corpus size divided by its occurrence frequency in the whole corpus. That is: $\text{IDF} = \log(\text{corpus_size}) / \text{document_frequency}$
- **TF-IDF Weight:** Assigns a term frequency/inverse document frequency score (**TF/IDF**) to the extracted n-grams. The value for each n-gram is its TF score multiplied by its IDF score.
- **Graph Weight:** Assigns score to the extracted n-grams based on the TextRank graph ranking. TextRank is a graph-based ranking model for text processing. Graph-based ranking algorithms are essentially a way of deciding importance based on global information. For more information, see [TextRank: Bringing Order into Texts](#) by Rada Mihalcea and Paul Tarau.

6. For all other options, see the property descriptions in the [preceding section](#).

7. Run the experiment.

See [this section](#) for an explanation of the results and their format.

Score or publish a model that uses n-grams

1. Copy the **Extract N-Gram Features from Text** module from the training dataflow to the scoring dataflow.
2. Connect the **Result Vocabulary** output from the training dataflow to the **Input Vocabulary** on the scoring dataflow.
3. In the scoring workflow, modify the **Extract N-Gram Features from Text** module and make these changes, leaving all else the same:
 - Set the **Vocabulary mode** parameter to **ReadOnly**.
 - Change the option **Use filter based feature selection** to **False**.
4. To publish the experiment, save the **Result Vocabulary** as dataset.

Then, connect the saved dataset to the **Extract N-Gram Features from Text** module in your scoring graph.

Results

The **Extract N-Gram Features from Text** module creates two types of output:

- **Results dataset:** A summary of the analyzed text together with the n-grams that were extracted. Columns that you did *not* select in the **Text column** option are passed through to the output. For each column of text that you analyze, the module generates these columns:
 - **NgramsString:** A string containing all unique n-grams.
 - **NumUniqueNgrams:** The count of n-grams extracted using the specified properties.
 - **Sparse matrix of n-gram occurrences:** The module generates a column for each n-gram found in the total corpus and adds a score in each column to indicate the weight of the n-gram for that row.
- **Result vocabulary:** The vocabulary contains the actual n-gram dictionary, together with the term frequency scores that are generated as part of the analysis. You can save the dataset for re-use with a different set of inputs, or for later update. You can also update the scores, or reuse the vocabulary for modeling and scoring.

Sample results

To illustrate how you can use the results, the following short example uses the Amazon Book Review dataset available in Studio (classic). The dataset was filtered to show only reviews with a score of 4 or 5, and reviews with a string length of under 300 characters.

From this dataset, a short review was selected, containing only 92 words. Here the author's name has been replaced with `xxx` and the book title replaced with `yyy`:

```
"Xxx at his best ! Yyy is one of Xxx's best yet! I highly recommend this novel."
```

Results dataset for sample review text

For this sample, the module generated these columns:

- **NumUniqueNgrams:** For this 92 word review, using the default settings, 11 n-grams were extracted from the sample review.

When the n-gram length was increased to 3 and the skip-gram value set to 1, 15 n-grams were found.

When feature selection was applied to the default, no n-grams were extracted.

- **NgramsString:** With the default settings, these n-grams were returned:

```
["his","best","one","highly","recommend","this","novel","his_best","highly_recommend","recommend_this",
"this_novel"]
```

With an n-gram length of 3 and skip-gram value of 1, these n-grams were returned:

```
["his","best","one","highly","recommend","this","novel","his_best","highly_recommend","recommend_this",
"this_novel","best_one","one_best","highly_this","highly_recommend_this"]
```

- **Sparse matrix of n-gram occurrences**

For this particular review, the results included these columns:

REVIEWTEXT.[MANAGES]	REVIEWTEXT. [AND_HIGHLY]	REVIEWTEXT.[HIGHLY]	REVIEWTEXT. [HIGHLY_RECOMMEND]
0	0	0.301511	0.301511

TIP

If you have trouble viewing a particular column, attach the [Select Columns in Dataset](#) module to the output, and then use the search function to filter columns by name.

Result vocabulary for sample review text

The vocabulary contains the actual n-gram dictionary, together with the term frequency scores that are generated as part of the analysis. You can save the dataset for re-use with a different set of inputs, or for later update. The scores DF and IDF are generated regardless of other options. When you combine vocabularies, these stored values are used as input to the weighting function you choose.

- **Id:** An identifier generated for each unique n-gram.
- **Ngram:** The n-gram. Spaces or other word separators are replaced by the underscore character.
- **DF:** The term frequency score for the n-gram in the original corpus.
- **IDF:** The inverse document frequency score for the n-gram in the original corpus.

It is possible to manually update this dataset; however, be careful, as you can introduce errors. For example:

- An error is raised if the module finds duplicate rows with the same key in the input vocabulary. Be sure that no two rows in the vocabulary have the same word.
- The input schema of the vocabulary datasets must match exactly, including column names and column types.
- The ID column and DF score column must be of the integer type.
- The IDF column must be of type FLOAT (floating point).

Technical notes

We recommend that you experiment with different ranges of values for n-gram length, the number of skip-grams, and use of feature selection to determine the dimensionality of your text corpus and the optimum feature ratio.

For more information about n-grams and skip-grams, see these resources:

- [Automatic Evaluation of Summaries Using N-gram Co-Occurrence Statistics](#)

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input data
Input vocabulary	Data Table	Input vocabulary

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Minimum number of non-zero elements	Integer	>=1	Applies only when using the following method: Count Based	1	Specify the number of features to output (for CountBased method)
Text column	Column Selection		Required	StringFeature	Name or one-based index of text column
Vocabulary mode	Vocabulary Mode	Create ReadOnly Update Merge	Required	Create	Specify how the n-gram vocabulary should be created from the corpus
N-Grams size	Integer	>=1	Required	1	Indicate the maximum size of n-grams to create
K-Skip size	Integer	>=0	Required	0	Indicate the k-skip size

Name	Type	Range	Optional	Default	Description
Weighting function	Weighting Function	Binary Weight TF Weight IDF Weight TF-IDF Weight Graph Weight	Required	Binary Weight	Choose the weighting function to apply to each n-gram value
Minimum word length	Integer	>=1	Required	3	Specify the minimum length of words to include in n-grams
Maximum word length	Integer	>=2	Required	25	Specify the maximum length of words to include in n-grams
Minimum n-gram document absolute frequency	Float	>=1.0	Required	5.0	Minimum n-gram document absolute frequency
Maximum n-gram document ratio	Float	>=0.0001	Required	1.0	Maximum n-gram document ratio
Detect out-of-vocabulary rows	Boolean		Required	true	Detect rows that have words not in the n-gram vocabulary (OOV)
Mark begin-of-sentence	Boolean		Required	false	Indicate whether a begin-sentence mark should be added to n-grams
Normalize n-gram feature vectors	Boolean		Required		Normalize n-gram feature vectors. If true, then the n-gram feature vector is divided by its L2 norm.
Use filter-based feature selection	True False Type	True False	Required	True	Use filter-based feature selection to reduce dimensionality

Name	Type	Range	Optional	Default	Description
Feature scoring method	Scoring Method	Pearson Correlation Mutual Information Kendall Correlation Spearman Correlation Chi Squared Fisher Score Count Based	Applies only when the option Use filter-based feature selection is True	Fisher Score	Choose the method to use for scoring
Target column	Column Selection		Applies when using one of the following methods: Pearson Correlation Mutual Information Kendall Correlation Spearman Correlation Chi Squared Fisher Score		Specify the target column
Number of desired features	Integer	>=1	Applies when using one of the following methods: Pearson Correlation Mutual Information Kendall Correlation Spearman Correlation Chi Squared Fisher Score	1	Specify the number of features to output in results

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Extracted features
Result vocabulary	Data Table	Result vocabulary

See also

[Text Analytics](#)

[A-Z List of Machine Learning Modules](#)

Feature Hashing

3/10/2021 • 8 minutes to read • [Edit Online](#)

Converts text data to integer encoded features using the Vowpal Wabbit library

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Feature Hashing** module in Azure Machine Learning Studio (classic), to transform a stream of English text into a set of features represented as integers. You can then pass this hashed feature set to a machine learning algorithm to train a text analysis model.

The feature hashing functionality provided in this module is based on the Vowpal Wabbit framework. For more information, see [Train Vowpal Wabbit 7-4 Model](#) or [Train Vowpal Wabbit 7-10 Model](#).

More about feature hashing

Feature hashing works by converting unique tokens into integers. It operates on the exact strings that you provide as input and does not perform any linguistic analysis or preprocessing.

For example, take a set of simple sentences like these, followed by a sentiment score. Assume that you want to use this text to build a model.

USERTEXT	SENTIMENT
I loved this book	3
I hated this book	1
This book was great	3
I love books	2

Internally, the **Feature Hashing** module creates a dictionary of n-grams. For example, the list of bigrams for this dataset would be something like this:

TERM (BIGRAMS)	FREQUENCY
This book	3
I loved	1
I hated	1

TERM (BIGRAMS)	FREQUENCY
I love	1

You can control the size of the n-grams by using the **N-grams** property. If you choose bigrams, unigrams are also computed. Thus, the dictionary would also include single terms like these:

TERM (UNIGRAMS)	FREQUENCY
book	3
I	3
books	1
was	1

After the dictionary has been built, the **Feature Hashing** module converts the dictionary terms into hash values, and computes whether a feature was used in each case. For each row of text data, the module outputs a set of columns, one column for each hashed feature.

For example, after hashing, the feature columns might look something like this:

RATING	HASHING FEATURE 1	HASHING FEATURE 2	HASHING FEATURE 3
4	1	1	0
5	0	0	0

- If the value in the column is 0, the row did not contain the hashed feature.
- If the value is 1, the row did contain the feature.

The advantage of using feature hashing is that you can represent text documents of variable-length as numeric feature vectors of equal-length, and achieve dimensionality reduction. In contrast, if you tried to use the text column for training as is, it would be treated as a categorical feature column, with many, many distinct values.

Having the outputs as numeric also makes it possible to use many different machine learning methods with the data, including classification, clustering, or information retrieval. Because lookup operations can use integer hashes rather than string comparisons, getting the feature weights is also much faster.

How to configure Feature Hashing

1. Add the **Feature Hashing** module to your experiment in Studio (classic).
2. Connect the dataset that contains the text you want to analyze.

TIP

Because feature hashing does not perform lexical operations such as stemming or truncation, you can sometimes get better results by doing text preprocessing before applying feature hashing. For suggestions, see the [Best practices](#) and [Technical notes](#) sections.

3. For **Target columns**, select those text columns that you want to convert to hashed features.

- The columns must be the string data type, and must be marked as a **Feature** column.

- If you choose multiple text columns to use as inputs, it can have a huge effect on feature dimensionality. For example, if a 10-bit hash is used for a single text column, the output contains 1024 columns. If a 10-bit hash is used for two text columns, the output contains 2048 columns.

NOTE

By default, Studio (classic) marks most text columns as features, so if you select all text columns, you might get too many columns, including many that are not actually free text. Use the **Clear feature** option in [Edit Metadata](#) to prevent other text columns from being hashed.

- Use **Hashing bitsize** to specify the number of bits to use when creating the hash table.

The default bit size is 10. For many problems, this value is more than adequate, but whether suffices for your data depends on the size of the n-grams vocabulary in the training text. With a large vocabulary, more space might be needed to avoid collisions.

We recommend that you try using a different number of bits for this parameter, and evaluate the performance of the machine learning solution.

- For **N-grams**, type a number that defines the maximum length of the n-grams to add to the training dictionary. An n-gram is a sequence of n words, treated as a unique unit.
- N-grams** = 1: Unigrams, or single words.
 - N-grams** = 2: Bigrams, or two-word sequences, plus unigrams.
 - N-grams** = 3: Trigrams, or three-word sequences, plus bigrams and unigrams.

- Run the experiment.

Results

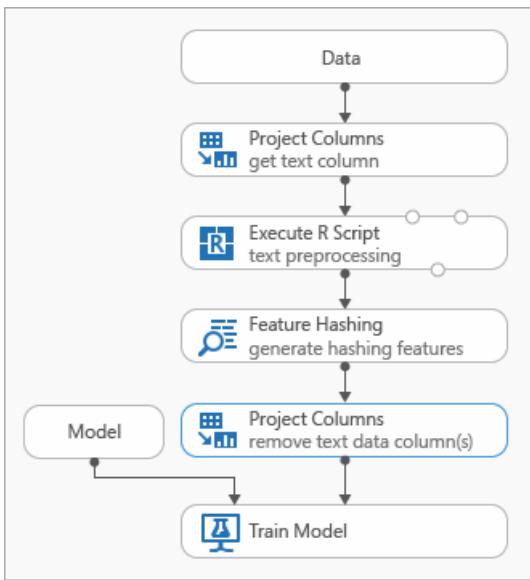
After processing is complete, the module outputs a transformed dataset in which the original text column has been converted to multiple columns, each representing a feature in the text. Depending on how big the dictionary is, the resulting dataset can be extremely large:

COLUMN NAME 1	COLUMN TYPE 2
USERTEXT	Original data column
SENTIMENT	Original data column
USERTEXT - Hashing feature 1	Hashed feature column
USERTEXT - Hashing feature 2	Hashed feature column
USERTEXT - Hashing feature n	Hashed feature column
USERTEXT - Hashing feature 1024	Hashed feature column

After you have created the transformed dataset, you can use it as the input to the [Train Model](#) module, together with a good classification model, such as [Two-Class Support Vector Machine](#).

Best practices

Some best practices that you can use while modeling text data are demonstrated in the following diagram representing an experiment



- You might need to add an [Execute R Script](#) module before using **Feature Hashing**, in order to preprocess the input text. With R script, you also have the flexibility to use custom vocabularies or custom transformations.
- You should add a [Select Columns in Dataset](#) module after the **Feature Hashing** module to remove the text columns from the output data set. You do not need the text columns after the hashing features have been generated.

Alternatively, you can use the [Edit Metadata](#) module to clear the feature attribute from the text column.

Also consider using these text preprocessing options, to simplify results and improve accuracy:

- word breaking
- stop word removal
- case normalization
- removal of punctuation and special characters
- stemming.

The optimal set of preprocessing methods to apply in any individual solution depends on domain, vocabulary, and business need. We recommend that you experiment with your data to see which custom text processing methods are most effective.

Examples

For examples of how feature hashing is used for text analytics, see the [Azure AI Gallery](#):

- [News Categorization](#): Uses feature hashing to classify articles into a predefined list of categories.
- [Similar Companies](#): Uses the text of Wikipedia articles to categorize companies.
- [Text Classification](#): This five-part sample uses text from Twitter messages to perform sentiment analysis.

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

TIP

In addition to using feature hashing, you might want to use other methods to extract features from text. For example:

- Use the **Preprocess Text** module to remove artifacts such as spelling errors, or to simplify text preparatory to hashing.
- Use **Extract Key Phrases** to use natural language processing to extract phrases.
- Use **Named Entity Recognition** to identify important entities.

Azure Machine Learning Studio (classic) provides a [Text Classification template](#) that guides you through using the **Feature Hashing** module for feature extraction.

Implementation details

The **Feature Hashing** module uses a fast machine learning framework called Vowpal Wabbit that hashes feature words into in-memory indexes, using a popular open source hash function called **murmurhash3**. This hash function is a non-cryptographic hashing algorithm that maps text inputs to integers, and is popular because it performs well in a random distribution of keys. Unlike cryptographic hash functions, it can be easily reversed by an adversary, so that it is unsuitable for cryptographic purposes.

The purpose of hashing is to convert variable-length text documents into equal-length numeric feature vectors, to support dimensionality reduction and make the lookup of feature weights faster.

Each hashing feature represents one or more n-gram text features (unigrams or individual words, bi-grams, tri-grams, etc.), depending on the number of bits (represented as k) and on the number of n-grams specified as parameters. It projects feature names to the machine architecture unsigned word using the murmurhash v3 (32-bit only) algorithm which then is AND-ed with $(2^k)-1$. That is, the hashed value is projected down to the first k lower-order bits, and the remaining bits are zeroed out. If the specified number of bits is 14, the hash table can hold $2^{14}-1$ (or 16,383) entries.

For many problems, the default hash table (bitsize = 10) is more than adequate; however, depending on the size of the n-grams vocabulary in the training text, more space might be needed to avoid collisions. We recommend that you try using a different number of bits for the **Hashing bitsize** parameter, and evaluate the performance of the machine learning solution.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Target columns	Any	ColumnSelection	StringFeature	Choose the columns to which hashing will be applied.
Hashing bitsize	[1;31]	Integer	10	Type the number of bits to use when hashing the selected columns

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
N-grams	[0;10]	Integer	2	Specify the number of N-grams generated during hashing. By default, both unigrams and bigrams are extracted

Outputs

NAME	TYPE	DESCRIPTION
Transformed dataset	Data Table	Output dataset with hashed columns

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Text Analytics](#)

[A-Z Module List](#)

Latent Dirichlet Allocation

3/10/2021 • 14 minutes to read • [Edit Online](#)

Use the Vowpal Wabbit library to perform VW LDA

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Latent Dirichlet Allocation** module in Azure Machine Learning Studio (classic), to group otherwise unclassified text into a number of categories. Latent Dirichlet Allocation (LDA) is often used in natural language processing (NLP) to find texts that are similar. Another common term is *topic modeling*.

This module takes a column of text, and generates these outputs:

- The source text, together with a score for each category
- A feature matrix, containing extracted terms and coefficients for each category
- A transformation, which you can save and reapply to new text used as input

Because this module uses the Vowpal Wabbit library, it is very fast. For more information about Vowpal Wabbit, see the [GitHub repository](#) which includes tutorials and an explanation of the algorithm.

More about Latent Dirichlet Allocation (LDA)

Generally speaking, LDA is not a method for classification per se, but uses a generative approach. What this means is that you don't need to provide known class labels and then infer the patterns. Instead, the algorithm generates a probabilistic model that is used to identify groups of topics. You can use the probabilistic model to classify either existing training cases, or new cases that you provide to the model as input.

A generative model can be preferable because it avoids making any strong assumptions about the relationship between the text and categories, and uses only the distribution of words to mathematically model topics.

- The theory is discussed in this paper, available as a PDF download: [Latent Dirichlet Allocation: Blei, Ng, and Jordan](#)
- The implementation in this module is based on the [Vowpal Wabbit library](#) (version 8) for LDA.

For more information, see the [Technical notes](#) section.

How to configure Latent Dirichlet Allocation

This module requires a dataset that contains a column of text, either raw or preprocessed.

1. Add the **Latent Dirichlet Allocation** module to your experiment.

2. As input for the module, provide a dataset containing one or more text columns.

3. For **Target columns**, choose one or more columns containing text to analyze.

You can choose multiple columns but they must be of the string data type.

In general, because LDA creates a large feature matrix from the text, you'll typically analyze a single text column.

4. For **Number of topics to model**, type an integer between 1 and 1000 that indicates how many categories or topics you want to derive from the input text.

By default, 5 topics are created.

5. For **N-grams**, specify the maximum length of N-grams generated during hashing.

The default is 2, meaning that both bigrams and unigrams are generated.

6. Select the **Normalize** option to converting output values to probabilities. Therefore, rather than representing the transformed values as integers, values in the output and feature dataset would be transformed as follows:

- Values in the dataset will be represented as a probability where $P(\text{topic}|\text{document})$.
- Values in the feature topic matrix will be represented as a probability where $P(\text{word}|\text{topic})$.

7. Select the option, **Show all options**, and then set it to TRUE if you want to view and then set additional advanced parameters.

These parameters are specific to the Vowpal Wabbit implementation of LDA. There are some good tutorials about LDA in Vowpal Wabbit online, as well as the official [Vowpal Wabbit Wiki](#).

See [this sample](#) for examples in version 8 and use of VW in Azure ML.

- **Rho parameter.** Provide a prior probability for the sparsity of topic distributions. Corresponds to VW's `lda_rho` parameter. You would use the value 1 if you expect that the distribution of words is flat; i.e, all words are assumed equiprobable. If you think most words appear sparsely, you might set it to a much lower value.
- **Alpha parameter.** Specify a prior probability for the sparsity of per-document topic weights. Corresponds to VW's `lda_alpha` parameter.
- **Estimated number of documents.** Type a number that represents your best estimate of the number of documents (rows) that will be processed. This lets the module allocate a hash table of sufficient size. Corresponds to the `lda_D` parameter in Vowpal Wabbit.
- **Size of the batch.** Type a number that indicates how many rows to include in each batch of text sent to Vowpal Wabbit. Corresponds to the `batch_sz` parameter in Vowpal Wabbit.
- **Initial value of iteration used in learning update schedule.** Specify the starting value for the learning rate. Corresponds to the `initial_t` parameter in Vowpal Wabbit.
- **Power applied to the iteration during updates.** Indicate the level of power applied to the iteration count during online updates. Corresponds to the `power_t` parameter in Vowpal Wabbit.
- **Number of passes over the data.** Specify the number of times the algorithm will cycle over the data. Corresponds to the `epoch_size` parameter in Vowpal Wabbit.

8. Select the option, **Build dictionary of ngrams** or **Build dictionary of ngrams prior to LDA**, if you want to create the n-gram list in an initial pass, before classifying text.

If you create the initial dictionary beforehand, you can later use the dictionary when reviewing the model. Being able to map results to text rather than numerical indices is generally easier for interpretation. However, saving the dictionary will take longer and use additional storage.

9. For **Maximum size of ngram dictionary**, type the total number of rows that can be created in the n-gram dictionary.

This option is useful for controlling the size of the dictionary. However, if the number of ngrams in the input exceeds this size, collisions may occur.

10. Run the experiment. The LDA module uses Bayes theorem to determine what topics might be associated with individual words. Words are not exclusively associated with any topics or groups; instead, each n-gram has a learned probability of being associated with any of the discovered classes.

Results

The module has two outputs:

- **Transformed dataset:** Contains the input text, and a specified number of discovered categories, together with the scores for each text example for each category.
- **Feature topic matrix:** The leftmost column contains the extracted text feature, and there is a column for each category containing the score for that feature in that category.

For details, see [Example of LDA results](#).

LDA transformation

This module also outputs the *transformation* that applies LDA to the dataset, as an [ITransform interface](#).

You can save this transformation and re-use it for other datasets. This might be useful if you have trained on a large corpus and want to reuse the coefficients or categories.

Refining an LDA model or results

Typically you cannot create a single LDA model that will meet all needs, and even a model designed for one task might require many iterations to improve accuracy. We recommend that you try all these methods to improve your model:

- Changing the model parameters
- Using visualization to understand the results
- Getting the feedback of subject matter experts to ascertain whether the generated topics are useful.

Qualitative measures can also be useful for assessing the results. To evaluate topic modeling results, consider:

- Accuracy - Are similar items really similar?
- Diversity - Can the model discriminate between similar items when required for the business problem?
- Scalability - Does it work on a wide range of text categories or only on a narrow target domain?

The accuracy of models based on LDA can often be improved by using natural language processing to clean, summarize and simplify, or categorize text. For example, the following techniques, all supported in Azure Machine Learning, can improve classification accuracy:

- Stop word removal
- Case normalization
- Lemmatization or stemming
- Named entity recognition

For more information, see [Preprocess Text](#) and [Named Entity Recognition](#).

In Studio (classic), you can also use R or Python libraries for text processing: [Execute R Script](#), [Execute Python Script](#)

Examples

For examples of text analytics, see these experiments in the [Azure AI Gallery](#):

- [Execute Python Script](#): Uses natural language processing in Python to clean and transform text.

For details and an example based on customer review text, see [Understanding LDA Results](#).

Example of LDA results

To illustrate how the **Latent Dirichlet Allocation** module works, the following example applies LDA with the default settings to the Book Review dataset provided in Azure Machine Learning Studio (classic).

Source dataset

The dataset contains a rating column, as well as the full comment text provided by users.

This table shows only a few representative examples.

TEXT
This book has its good points. If anything, it helps you put into words what you want from a supervisor....
I admit, I haven't finished this book. A friend recommended it to me as I have been having problems with insomnia...
Poorly written I tried reading this book but found it so turgid and poorly written that I put it down in frustration. ...
Since borrowing a dog-eared copy from friends who were passing it around a number of years ago, I have not been able to get my hands on this book which became a short-lived cult favorite
The plot of this book was interesting, and it could have been a good book. Unfortunately, it wasn't. The main problem for me was that ...

During processing, the **Latent Dirichlet Allocation** module both cleans and analyzes the text, based on parameters you specify. For example, it can automatically tokenize the text and remove punctuation, and at the same time find the text features for each topic.

LDA transformed dataset

The following table contains the **transformed** dataset, based on the Book Review sample. The output contains the input text, and a specified number of discovered categories, together with the scores for each category.

MOVIE NAME	TOPIC 1	TOPIC 2	TOPIC 3	TOPIC 4	TOPIC 5
this book has its good points	0.001652892	0.001652892	0.001652892	0.001652892	0.9933884
friend recommended it to me	0.00198019	0.001980198	0.9920791	0.001980198	0.001980198
tried reading this book	0.002469135	0.002469135	0.9901233	0.002469135	0.002469135
borrowed it from friend	0.9901232	0.002469135	0.002469135	0.002469135	0.002469135

MOVIE NAME	TOPIC 1	TOPIC 2	TOPIC 3	TOPIC 4	TOPIC 5
plot of this book was interesting	0.001652892	0.001652892	0.9933884	0.001652892	0.001652892

In this example, we used the default value of 5 for **Number of topics to model**. Therefore, the LDA module creates five categories, which we can assume will correspond roughly with the original five-scale rating system.

The module also assigns a score to each entry for each of the five categories that represent topics. A score indicates the probability that the row should be assigned to a particular category.

Feature topic matrix

The second output of the module is the **feature topic matrix**. This is a tabular dataset that contains the *featurized text*, in column **Feature**, along with a score for each of the categories, in the remaining columns *Topic 1, Topic 2, ... Topic N*. The score represents the coefficient.

FEATURE	TOPIC 1	TOPIC 2	TOPIC 3	TOPIC 4	TOPIC 5
interesting	0.02402820719 83144	0.03546789547 79375	0.36305186657 6914	0.02766378243 15893	0.66066357614 9515
was	0.01714787295 32397	0.08239690311 08669	0.00452966877 950789	0.04087145103 19233	0.02507732268 9733
from	0.01482242203 49217	0.05050869814 92109	0.00434423322 461094	0.02733891262 93824	0.01714843551 06826
plot	0.02274158893 48212	0.04087094564 89325	0.18279104134 5191	0.08693709081 2819	1 0.01696801367 08971
reading	0.02274158893 48212	0.04087094564 89325	0.18279104134 5191	0.08693709081 28191	0.01696801367 08971
tried	0.02697249791 47211	0.03902626355 1767	0.00443749106 785087	0.06288298160 88284	0.02353407288 18033
me	0.02626569451 40134	0.03669413027 51921	0.00656837975 179138	0.03292145761 60066	0.02141218511 06808
to	0.01410261032 24462	0.04335997691 9215	0.00388640531 859447	0.03059259534 40055	0.02289937505 26364
it	0.02644905471 05951	0.03566744403 11847	0.00541759897 864314	0.03145393862 50293	0.01406064685 87681
friend	0.01359713229 60941	0.03461181714 67234	0.00434999437 350706	0.06665073218 88536	0.01815686377 9311
points	0.02274158893 48212	0.03962338557 19081	0.00404663601 474112	0.03811565100 19025	0.03377880094 96797
good	0.65181307383 6783	0.05986463974 44108	0.00446809691 985617	0.03589756946 46062	0.01389891244 11206

FEATURE	TOPIC 1	TOPIC 2	TOPIC 3	TOPIC 4	TOPIC 5
its	0.01853855886 47078	0.14425398678 3184	0.00408876416 453866	0.05830492404 41475	0.01544280556 6858
of	0.01714167802 45647	0.05593611804 18586	0.01006339045 44953	0.08709393010 6723	0.01825738338 69842
borrowed	0.01714167802 45647	0.05593611804 18586	0.01006339045 44953	0.08709393010 6723	0.01825738338 69842
has	0.01714167802 45647	0.05593611804 18586	0.01006339045 44953	0.08709393010 6723	0.01825738338 69842
book	0.01431570479 20681	0.06914594853 5052	0.18403634017 0983	0.05487573378 23903	0.01568379769 85903
recommended	0.01614868484 19689	0.03991433263 99534	0.00550113530 229642	0.02863714914 2764	0.01476751390 39372
this	0.01614868484 19689	0.03991433263 99534	0.00550113530 229642	0.02863714914 2764	0.01476751390 39372

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Implementation details

By default, the distributions of outputs for transformed dataset and feature-topic matrix are normalized as probabilities.

- The transformed dataset is normalized as the conditional probability of topics given a document. In this case, the sum of each row equals 1.
- The feature-topic matrix is normalized as the conditional probability of words given a topic. In this case, the sum of each column equals 1.

TIP

Occasionally the module might return an empty topic, which is most often caused by the pseudo-random initialization of the algorithm. If this happens, you can try changing related parameters, such as the maximum size of the N-gram dictionary or the number of bits to use for feature hashing.

LDA and topic modeling

Latent Dirichlet Allocation (LDA) is often used for *content-based topic modeling*, which basically means learning categories from unclassified text. In content-based topic modeling, a topic is a distribution over words.

For example, assume that you have provided a corpus of customer reviews that includes many, many products. The text of reviews that have been submitted by many customers over time would contain many terms, some of which are used in multiple topics.

A **topic** that is identified by the LDA process might represent reviews for an individual Product A, or it might represent a group of product reviews. To LDA, the topic itself is just a probability distribution over time for a set of words.

Terms are rarely exclusive to any one product, but can refer to other products, or be general terms that apply to everything ("great", "awful"). Other terms might be noise words. However, it is important to understand that the LDA method does not purport to capture all words in the universe, or to understand how words are related, aside from probabilities of co-occurrence. It can only group words that were used in the target domain.

After the term indexes have been computed, individual rows of text are compared using a distance-based similarity measure, to determine whether two pieces of text are like each other. For example, you might find that the product has multiple names that are strongly correlated. Or, you might find that strongly negative terms are usually associated with a particular product. You can use the similarity measure both to identify related terms and to create recommendations.

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input dataset

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Number of hash bits	Integer	[1;31]	Applies when the Show all options checkbox is <i>not</i> selected	12	Number of bits to use for feature hashing
Target column(s)	Column Selection		Required	StringFeature	Target column name or index
Number of topics to model	Integer	[1;1000]	Required	5	Model the document distribution against N topics
N-grams	Integer	[1;10]	Required	2	Order of N-grams generated during hashing
Normalize	Boolean		Required	true	Normalize output to probabilities. The transformed dataset will be $P(\text{topic} \text{document})$ and the feature topic matrix will be $P(\text{word} \text{topic})$.
Show all options	Boolean	True or False	Required	False	Presents additional parameters specific to Vowpal Wabbit online LDA

Name	Type	Range	Optional	Default	Description
Rho parameter	Float	[0.00001;1.0]	Applies when the Show all options checkbox is selected	0.01	Rho parameter
Alpha parameter	Float	[0.00001;1.0]	Applies when the Show all options checkbox is selected	0.01	Alpha parameter
Estimated number of documents	Integer	[1;int.MaxValue]	Applies when the Show all options checkbox is selected	1000	Estimated number of documents (Corresponds to lda_D parameter)
Size of the batch	Integer	[1;1024]	Applies when the Show all options checkbox is selected	32	Size of the batch
Initial value of iteration used in learning rate update schedule	Integer	[0;int.MaxValue]	Applies when the Show all options checkbox is selected	0	Initial value of iteration count used in learning rate update schedule (Corresponds to initial_t parameter)
Power applied to the iteration during updates	Float	[0.0;1.0]	Applies when the Show all options checkbox is selected	0.5	Power applied to the iteration count during online updates (Corresponds to power_t parameter)
Number of training iterations	Integer	[1;1024]	Applies when the Show all options checkbox is selected	25	Number of training iterations
Build dictionary of ngrams	Boolean	True or False	Applies when the Show all options checkbox is <i>not</i> selected	True	Builds a dictionary of ngrams prior to computing LDA. Useful for model inspection and interpretation

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Number of bits to use for feature hashing	Integer	[1;31]	Applies when the option Build dictionary of ngrams is False	12	Number of bits to use during feature hashing
Maximum size of ngram dictionary	Integer	[1;int.MaxValue]	Applies when the option Build dictionary of ngrams is True	20000	Maximum size of the ngrams dictionary. If number of tokens in the input exceed this size, collisions may occur
Build dictionary of ngrams prior to LDA	Boolean	True or False	Applies when the Show all options checkbox is selected	True	Builds a dictionary of ngrams prior to LDA. Useful for model inspection and interpretation
Maximum number of ngrams in dictionary	Integer	[1;int.MaxValue]	Applies when the option Build dictionary of ngrams is True and the Show all options checkbox is selected	20000	Maximum size of the dictionary. If number of tokens in the input exceed this size, collisions may occur

Outputs

NAME	TYPE	DESCRIPTION
Transformed dataset	Data Table	Output dataset
Feature topic matrix	Data Table	Feature topic matrix produced by LDA
LDA transformation	ITransform interface	Transformation that applies LDA to the dataset

Exceptions

EXCEPTION	DESCRIPTION
Error 0002	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.

EXCEPTION	DESCRIPTION
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Text Analytics](#)

[Feature Hashing](#)

[Named Entity Recognition](#)

[Score Vowpal Wabbit 7-4 Model](#)

[Train Vowpal Wabbit 7-4 Model](#)

[Train Vowpal Wabbit 8 Model](#)

Named Entity Recognition

3/10/2021 • 5 minutes to read • [Edit Online](#)

Recognizes named entities in a text column

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Named Entity Recognition** module in Azure Machine Learning Studio (classic), to identify the names of things, such as people, companies, or locations in a column of text.

Named entity recognition is an important area of research in machine learning and natural language processing (NLP), because it can be used to answer many real-world questions, such as:

- Does a tweet contain the name of a person? Does the tweet also provide his current location?
- Which companies were mentioned in a news article?
- Were specified products mentioned in complaints or reviews?

To get a list of named entities, you provide a dataset as input that contains a text column. The **Named Entity Recognition** module will then identify three types of entities: people (PER), locations (LOC), and organizations (ORG).

The module also labels the sequences by where these words were found, so that you can use the terms in further analysis.

For example, the following table shows a simple input sentence, and the terms and values generated by the module:

INPUT TEXT	MODULE OUTPUT
"Boston is a great place to live."	0,Boston,0,6,LOC

The output can be interpreted as follows:

- The first '0' means that this string is the first article input to the module.

Because a single article can have multiple entities, including the article row number in the output is important for mapping features to articles.

- `Boston` is the recognized entity.
- The `0` that follows `Boston` means the entity `Boston` starts from the first letter of the input string. Indices are zero-based.

- `6` means the length of the entity `Boston` is 6.
- `Loc` means the entity `Boston` is a place, or location. Other supported named entity types are person (`PER`) and organization (`ORG`).

How to configure Named Entity Recognition

1. Add the **Named Entity Recognition** module to your experiment in Studio (classic). You can find the module in the **Text Analytics** category.
2. On the input named **Story**, connect a dataset containing the text to analyze.

The "story" should contain the text from which to extract named entities.

The column used as **Story** should contain multiple rows, where each row consists of a string. the string can be short, like a sentence, or long, like a news article.

You can connect any dataset that contains a text column. However, if the input dataset contains multiple columns, use [Select Columns in Dataset](#) to choose only the column that contains the text you want to analyze

NOTE

The second input, **Custom Resources (Zip)**, is not supported at this time.

In future, you can add custom resource files here, for identifying different entity types.

3. Run the experiment.

Results

The module outputs a dataset containing a row for each entity that was recognized, together with the offsets.

Because each row of input text might contain multiple named entities, an article ID number is automatically generated and included in the output, to identify the input row that contained the named entity. The article ID is based on the natural order of the rows in the input dataset.

You can convert this output dataset to CSV for download or save it as a dataset for re-use.

Use named entity recognition in a web service

If you publish a web service from Azure Machine Learning Studio (classic) and want to consume the web service by using C#, Python, or another language such as R, you must first implement the service code provided on the help page of the web service.

If your web service provides multiple rows of output, the URL of the web service that you add to your C#, Python, or R code should have the suffix `scorermultirow` instead of `score`.

For example, assume you use the following URL for your web service:

```
https://ussouthcentral.services.azureml.net/workspaces/<workspace id>/services/<service id>/score
```

To enable multi-row output, change the URL to

```
https://ussouthcentral.services.azureml.net/workspaces/<workspace id>/services/<service id>/scorermultirow
```

To publish this web service, you should add an additional [Execute R Script](#) module after the [Named Entity Recognition](#) module, to transform the multi-row output into a single delimited with semi-colons (;). The reason for consolidating the multiple rows of output into a single row is to return multiple entities per input row.

For example, let's assume you have an input sentence with two named entities. Rather than returning two rows

for each row of input, you can return a single rows with multiple entities, separated by semi-colons as shown here:

INPUT TEXT	OUTPUT OF WEB SERVICE
Microsoft has two office locations in Boston.	0,Microsoft,0,9,ORG,;,0,Boston,38,6,LOC,;

The following code sample demonstrates how to do this:

```
# Map 1-based optional input ports to variables
d <- maml.mapInputPort(1) # class: data.frame
y=length(d) ##size of cols
x=dim(d)[1] ##size of rows
longd=matrix("NA",nrow=1,ncol=x*(y+1))
for (i in 1:x)
{
  for (j in 1:y)
  {
    longd[1,j+(i-1)*(y+1)]=toString(d[i,j])
  }
  longd[1,j+(i-1)*(y+1)+1]=c(";")
}

final_output=as.data.frame(longd)
# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("final_output");
```

Examples

This blog provides an extended explanation of how named entity recognition works, its background, and possible applications:

- [Machine learning and text analytics](#)

Also, see the following sample experiments in the [Azure AI Gallery](#) for demonstrations of how to use text classification methods commonly used in machine learning:

- [News Categorization sample](#): Uses feature hashing to classify articles into a predefined list of categories.
- [Similar Companies sample](#): Uses the text of Wikipedia articles to categorize companies.
- [Text-Classification Step 1 of 5: Data preparation](#): In this five-part walkthrough of text classification, text from Twitter messages is used to perform sentiment analysis. A variety of text pre-processing techniques are also demonstrated.

Technical notes

Language support

Currently, the **Named Entity Recognition** module supports only English text. It can detect organization names, personal names, and locations in English sentences. If you use the module on other languages, you might not get an error, but the results are not as good as for English text.

In future, support for additional languages can be enabled by integrating the multilingual components provided in the Office Natural Language Toolkit.

Expected inputs

NAME	TYPE	DESCRIPTION
Story	Data Table	An input dataset (DataTable) that contains the text column you want to analyze.
CustomResources	Zip	(Optional) A file in ZIP format that contains additional custom resources. This option is not available currently and is provided for forward compatibility only.

Outputs

NAME	TYPE	DESCRIPTION
Entities	Data Table	A list of character offsets and entities

See also

[Text Analytics](#)
[Feature Hashing](#)
[Score Vowpal Wabbit 7-4 Model](#)
[Train Vowpal Wabbit 7-4 Model](#)

Preprocess Text

3/10/2021 • 12 minutes to read • [Edit Online](#)

Performs cleaning operations on text

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Preprocess Text** module in Azure Machine Learning Studio (classic), to clean and simplify text. By preprocessing the text, you can more easily create meaningful features from text.

For example, the **Preprocess Text** module supports these common operations on text:

- Removal of stop-words
- Using regular expressions to search for and replace specific target strings
- Lemmatization, which converts multiple related words to a single canonical form
- Filtering on specific parts of speech
- Case normalization
- Removal of certain classes of characters, such as numbers, special characters, and sequences of repeated characters such as "aaaa"
- Identification and removal of emails and URLs

You can choose which cleaning options to use, and optionally specify a custom list of stop-words.

The module currently supports six languages: English, Spanish, French, Dutch, German and Italian.

How to configure Text Preprocessing

1. Add the **Preprocess Text** module to your experiment in Studio (classic). You can find this module under [Text Analytics](#).
2. Connect a dataset that has at least one column containing text.
3. If the text you are preprocessing is all in the same language, select the language from the **Language** dropdown list. With this option, the text is preprocessed using linguistic rules specific to the selected language.
4. To preprocess text that might contain multiple languages, choose the **Column contains language** option.

Then, use the **Culture-language column** property to choose a column in the dataset that indicates the language used in each row. The column must contain a standard language identifier, such as "English" or `en`.

Based on this identifier, the module applies appropriate linguistic resources to process the text.

If your dataset does not contain such identifiers, use the **Detect Language** module to analyze the language beforehand, and generate an identifier.

TIP

An error is raised if an unsupported language is included. See the [Technical notes](#) section for more information.

5. **Remove by part of speech:** Select this option if you want to apply part-of-speech analysis. You can then use the part-of-speech tags to remove certain classes of words.

- **Remove nouns:** Select this option to remove nouns.
- **Remove adjectives:** Select this option to remove adjectives.
- **Remove verbs:** Select this option to remove verbs.

For more information about the part-of-speech identification method used, see the [Technical notes](#) section.

6. **Text column to clean:** Select the column or columns that you want to preprocess.

7. **Remove stop words:** Select this option if you want to apply a predefined stopword list to the text column. Stop word removal is performed before any other processes.

Stopword lists are language dependent and customizable; for more information, see the [Technical notes](#) section.

8. **Lemmatization:** Select this option if you want words to be represented in their canonical form. This option is useful for reducing the number of unique occurrences of otherwise similar text tokens.

The lemmatization process is highly language-dependent; see the [Technical notes](#) section for details.

9. **Detect sentences:** Select this option if you want the module to insert a sentence boundary mark when performing analysis.

This module uses a series of three pipe characters `|||` to represent the sentence terminator.

10. Optionally, you can perform custom find-and-replace operations using regular expressions.

- **Custom regular expression:** Define the text you are searching for.
- **Custom replacement string:** Define a single replacement value.

11. **Normalize case to lowercase:** Select this option if you want to convert ASCII uppercase characters to their lowercase forms.

If characters are not normalized, the same word in uppercase and lowercase letters is considered two different words: for example, `AM` is the same as `am`.

12. Optionally, you can remove the following types of characters or character sequences from the processed output text:

- **Remove numbers:** Select this option to remove all numeric characters for the specified language. Identification of what constitutes a number is domain dependent and language dependent. If numeric characters are an integral part of a known word, the number might not be removed.
- **Remove special characters:** Use this option to replace any non-alphanumeric special characters with the pipe `|` character.

For more about special characters, see the [Technical notes](#) section.

- **Remove duplicate characters:** Select this option to remove any sequences that repeat

characters. For example, a sequence like "aaaaa" would be removed.

- **Remove email addresses:** Select this option to remove any sequence of the format `<string>@<string>`.
- **Remove URLs:** Select this option to remove any sequence that includes the following URL prefixes:
 - `http`, `https`
 - `ftp`
 - `www`

13. **Expand verb contractions:** This option applies only to languages that use verb contractions; currently, English only.

For example, by selecting this option, you could replace the phrase "*wouldn't stay there*" with "*would not stay there*".

14. **Normalize backslashes to slashes:** Select this option to map all instances of `\\" to /`.

15. **Split tokens on special characters:** Select this option if you want to break words on characters such as `&`, `-`, and so forth.

For example, the string `MS-WORD` would be separated into two tokens, `MS` and `WORD`.

Examples

The following examples in the [Azure AI Gallery](#) illustrate the use of the **Preprocess Text** module:

- [How to use a custom stopword list](#)
- [How to modify the default stopword list](#)

Technical notes

This section provides more information about the underlying text pre-processing technology, and how to specify custom text resources.

Supported languages

Currently Azure Machine Learning supports text preprocessing in these languages:

- Dutch
- English
- French
- German
- Italian
- Spanish

Additional languages are planned. See the [Microsoft Machine Learning blog](#) for announcements.

Lemmatization

Lemmatization is the process of identifying a single canonical form to represent multiple word tokens.

The natural language processing libraries included in Azure Machine Learning Studio (classic) combine the following multiple linguistic operations to provide lemmatization:

- **Sentence separation:** In free text used for sentiment analysis and other text analytics, sentences are frequently run-on or punctuation might be missing. Input texts might constitute an arbitrarily long chunk

of text, ranging from a tweet or fragment to a complete paragraph, or even document.

The natural language tools used by Studio (classic) perform sentence separation as part of the underlying lexical analysis. However, sentences are not separated in the output. Optionally, you can specify that a sentence boundary be marked to aid in other text processing and analysis.

- **Tokenization:** The rules that determine the boundaries of words are language-dependent and can be complex even in languages that use spaces between words.

Some languages (such as Chinese or Japanese) do not use any white space between words, and separation of words requires morphological analysis.

Therefore, the tokenization methods and rules used in this module provide different results from language to language. These tokenization rules are determined by text analysis libraries provided by Microsoft Research for each supported language, and cannot be customized.

- **Part-of-speech identification:** In any sequence of words, it can be difficult to computationally identify the exact part of speech for each word. Even an apparently simple sentence such as "Time flies like an arrow" can have many dozen parses (a famous example). Parts of speech are also very different depending on the morphology of different languages.

In Azure Machine Learning, a disambiguation model is used to choose the **single most likely part of speech**, given the current sentence context. The part-of-speech information is used to help filter words used as features and aid in key-phrase extraction. However, the output of this module does not explicitly include POS tags and therefore cannot be used to generate POS-tagged text.

- **Generating dictionary form:** A word may have multiple *lemmas*, or dictionary forms, each coming from a different analysis. For instance, the English word building has two possible lemmas: *building* if the word is a noun ("the tall building"), or *build* if the word is a verb ("they are building a house"). In Azure Machine Learning, only the *single most probable* dictionary form is generated.

Sample lemmatization output

SOURCE	LEMMATIZED WITH CASE CONVERSION
He is swimming	he i swim
He is going for a swim	he i go for a swim
Swimming is good for building muscle	swim be good for build muscle
He is building a building	he i build a build
We are all building buildings	we be all build building

NOTE

The language models used to generate dictionary form have been trained and tested against a variety of general purpose and technical texts, and are used in many other Microsoft products that require natural language APIs. However, natural language is inherently ambiguous and 100% accuracy on all vocabulary is not feasible. For example, lemmatization can be affected by other parts of speech, or by the way that the sentence is parsed.

If you need to perform additional pre-processing, or perform linguistic analysis using a specialized or domain-dependent vocabulary, we recommend that you use customizable NLP tools, such as those available in Python and R.

Special characters

Special characters are defined as single characters that cannot be identified as any other part of speech, and can include punctuation: colons, semi-colons, and so forth.

Stopwords

A **stop word** (or **stopword**) is a word that is often removed from indexes because it is common and provides little value for information retrieval, even though it might be linguistically meaningful.

For example, many languages make a semantic distinction between definite and indefinite articles ("the building" vs "a building"), but for machine learning and information retrieval, the information is sometimes not relevant. Hence you might decide to discard these words.

The Azure Machine Learning environment includes lists of the most common stopwords for each of the supported languages.

LANGUAGE	NUMBER OF STOPWORDS	EXAMPLES
Dutch	49	aan, af, al
English	312	a, about, above
French	154	de, des, d', la
German	602	a, ab, aber
Italian	135	a, adesso, ai
Spanish	368	ésa, ésta, éste

For your convenience, a zipped file containing the default stopwords for all current languages has been made available in Azure storage: [Stopwords.zip](#).

How to modify the stopword list

We expect that many users want to create their own stopword lists, or change the terms included in the default list. The following experiment in the [Cortana Intelligence Gallery](#) demonstrates how you can customize a stop word list.

- [How to modify the default stopword list](#)

If you modify the list, or create your own stop word list, observe these requirements:

- The file must contain a single text column. You might get the following error if an additional column is present: "Preprocess Text Error Column selection pattern "Text column to clean" is expected to provide 1 column(s) selected in input dataset, but 2 column(s) is/are actually provided. (Error 0022)"

If this happens, look for spaces, tabs, or hidden columns present in the file from which the stopword list was originally imported. Depending on how the file was prepared, tabs or commas included in text can also cause multiple columns to be created.

When you get this error, review the source file, or use the [Select Columns in Dataset](#) module to choose a single column to pass to the [Preprocess Text](#) module.

- Each row can contain only one word. For the purposes of parsing the file, words are determined by insertion of spaces.
- The stopword list cannot be empty.

Order of operations

In this module, you can apply multiple operations to text. However, the order in which these operations are applied cannot be changed. This can affect the expected results.

For example, if you apply lemmatization to text, and also use stopword removal, all the words are converted to their lemma forms before the stopword list is applied. Therefore, if your text includes a word that is not in the stopword list, but its lemma is in the stopword list, the word would be removed.

Be sure to test target terms in advance to guarantee the correct results.

Unsupported languages

If your text column includes languages not supported by Azure Machine Learning, we recommend that you use only those options that do not require language-dependent processing. This can help avoid strange results.

Also, if you use the option **Column contains language**, you must ensure that no unsupported languages are included in the text that is processed. If an unsupported language or its identifier is present in the dataset, the following run-time error is generated:

"Preprocess Text Error (0039): Please specify a supported language."

To avoid failing the entire experiment because an unsupported language was detected, use the [Split Data](#) module, and specify a regular expression to divide the dataset into supported and unsupported languages.

For example, the following regular expression splits the dataset based on the detected language for the column

Sentence :

```
\"Sentence Language" Dutch|English|French|Italian|Spanish
```

If you have a column that contains the language identifier, or if you have generated such a column, you can use a regular expression such as the following to filter on the identifier column:

```
\"Sentence Iso6391 Name" nl|en|fr|it|es
```

Expected inputs

NAME	TYPE	DESCRIPTION
Dataset	Data Table	Input data
Stop words	Data Table	Optional custom list of stop words to remove

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Remove URLs	Boolean	True False	Required	true	Remove URLs

Name	Type	Range	Optional	Default	Description
Language	Language	English Spanish French Dutch German Italian	Required	English	Select the language to preprocess
Text column to clean	Column Selection		Required	StringFeature	Select the text column to clean
Custom regular expression	String		Optional		Specify the custom regular expression
Custom replacement string	String		Optional		Specify the custom replacement string for the custom regular expression
Remove stop words	Boolean		Required	true	Remove stop words
Lemmatization	Boolean		Required	true	Use lemmatization
Remove by part of speech	True False Type	true false	Required	False	Indicate whether part-of-speech analysis should be used to identify and remove certain word classes
Remove nouns	Boolean		Applies when the Filter by part of speech option is selected	true	Remove nouns
Remove adjectives	Boolean		Applies when the Filter by part of speech option is selected	true	Remove adjectives
Remove verbs	Boolean		Applies when the Filter by part of speech option is selected	true	Remove verbs

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Detect sentences	Boolean		Required	true	Detect sentences by adding a sentence terminator \" \n" that can be used by the n-gram features extractor module
Normalize case to lowercase	Boolean		Required	true	Normalize case to lowercase
Remove numbers	Boolean		Required	true	Remove numbers
Remove special characters	Boolean		Required	true	Remove non-alphanumeric special characters and replace them with \"\ \" character
Remove duplicate characters	Boolean		Required	true	Remove duplicate characters
Remove email addresses	Boolean		Required	true	Remove email addresses

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Results dataset

Exceptions

EXCEPTION	DESCRIPTION
Error 0003	An exception occurs if one or more of inputs are null or empty.
Error 0030	an exception occurs in when it is not possible to download a file.
Error 0048	An exception occurs when it is not possible to open a file.
Error 0049	An exception occurs when it is not possible to parse a file.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Text Analytics](#)

[A-Z Module List](#)

Score Vowpal Wabbit Version 7-4 Model

3/10/2021 • 4 minutes to read • [Edit Online](#)

Scores data using the Vowpal Wabbit machine learning system from the command line interface

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Score Vowpal Wabbit Version 7-4 Model** module in Azure Machine Learning Studio (classic), to generate scores for a set of input data, using an existing trained Vowpal Wabbit model.

This module is provided for compatibility with version 7-4 of the Vowpal Wabbit framework. Use this module only if you need to score data using a trained model that was saved in the 7-4 format.

To create a new VW model, we recommend that you use the latest version::

- [Train Vowpal Wabbit 8 Model](#)
- [Score Vowpal Wabbit 8 Model](#)

How to configure Score Vowpal Wabbit Version 7-4 Model

1. Add the **Score Vowpal Wabbit Version 7-4 Model** module to your experiment.
2. Add a trained Vowpal Wabbit model and connect it to the left-hand input port. You can use a trained model created in the same experiment, or locate a saved model in the **Trained Models** group of Studio (classic)'s left navigation pane.

Restrictions

The model must be available in Azure Machine Learning Studio (classic); you cannot directly load a model from Azure storage.

Only Vowpal Wabbit 7-4 models are supported; you cannot connect saved models that were trained by using other algorithms, and you cannot use models that were trained using later versions.

3. In the **VW arguments** text box, type a set of valid command-line arguments to the Vowpal Wabbit executable.

For information about which Vowpal Wabbit arguments are supported in Azure Machine Learning, see the [Technical notes](#) section.

4. Click **Specify data type**, and select one of the supported data types from the list.

Scoring requires a single column of VW-compatible data.

If you have an existing file that was created in the SVMLight or VW formats, you can load it into the Azure

ML workspace as a new dataset in one of these formats: Generic CSV without header, TSV without header.

The **VW** option requires that a label be present, but it is not used in scoring except for comparison.

5. Add an [Import Data](#) module and connect it to the right-hand input port of **Score Vowpal Wabbit**

Version 7-4. Configure the [Import Data](#) module to access the input data.

The input data for scoring must have been prepared ahead of time in one of the supported formats and stored in Azure blob storage.

6. Select the option, **Include an extra column containing labels**, if you want to output labels together with the scores.

Typically, when handling text data, Vowpal Wabbit does not require labels, and will return only the scores for each row of data.

7. Select the option, **Use cached results**, if you want to re-use results from a previous run, assuming the following conditions are met:

- A valid cache exists from a previous run.
- The input data and parameters settings of the module have not changed since the previous run.

Otherwise, the import process is repeated each time the experiment runs.

8. Run the experiment.

Results

After training is complete:

- To visualize the results, right-click the output of the [Score Vowpal Wabbit Version 7-4 Model](#) module.

The output indicates a prediction score normalized from 0 to 1.

Examples

For examples of how Vowpal Wabbit can be used in machine learning, see the [Azure AI Gallery](#):

- [Vowpal Wabbit sample](#)

This experiment demonstrates data preparation, training, and operationalization of a VW model.

The following video provides a walkthrough of the training and scoring process for Vowpal Wabbit:

- [Text analytics and Vowpal Wabbit in Azure ML Studio \(classic\)](#)

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Supported and unsupported parameters

Vowpal Wabbit has many command-line options for choosing and tuning algorithms. A full discussion of these options is not possible here; we recommend that you view the [Vowpal Wabbit wiki page](#).

The following parameters are not supported in Azure Machine Learning Studio (classic).

- The input/output options specified in https://github.com/JohnLangford/vowpal_wabbit/wiki/Command-line-arguments

These properties are already configured automatically by the module.

- Additionally, any option that generates multiple outputs or takes multiple inputs is disallowed. These include `--cbt`, `--Lda`, and `--wap`.
- Only supervised learning algorithms are supported. This disallows these options: `-active`, `--rank`, `--search` etc.

All arguments other than those described above are allowed.

Expected inputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	Trained learner
Dataset	Data Table	Dataset to be scored

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
VW arguments	Any	String		Type Vowpal Wabbit arguments. The following arguments are not supported: - -i - -p or - -t
Include an extra column containing labels	Any	Boolean	false	Specify whether the zipped file should include labels with the predictions
Specify data type	VW SVMLight	DataType	VW	Indicate whether the file format is SVMLight or Vowpal Wabbit

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with the prediction results

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.

EXCEPTION	DESCRIPTION
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Text Analytics](#)

[Feature Hashing](#)

[Named Entity Recognition](#)

[Vowpal Wabbit Score](#)

[Train Vowpal Wabbit 7-4 Model](#)

[Train Vowpal Wabbit 7-10 Model](#)

[A-Z Module List](#)

Score Vowpal Wabbit Version 7-10 Model

3/10/2021 • 4 minutes to read • [Edit Online](#)

Scores data using the Vowpal Wabbit machine learning system from the command line interface

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Score Vowpal Wabbit Version 7-10 Model** module in Azure Machine Learning Studio (classic), to generate scores for a set of input data, using an existing Vowpal Wabbit model.

This module uses version 7-10 of the Vowpal Wabbit framework. Use this module to score data using a trained model that was saved in the 7-10 format.

If you have existing models created using an earlier version, use these modules:

- [Train Vowpal Wabbit 7-4 Model](#)
- [Score Vowpal Wabbit 7-4 Model](#)

For the latest version of Vowpal Wabbit, use:

- [Train Vowpal Wabbit 8 Model](#)
- [Score Vowpal Wabbit 8 Model](#)

How to configure Score Vowpal Wabbit Version 7-10 Model

1. Add the **Score Vowpal Wabbit Version 7-10 Model** module to your experiment.
2. Add a trained Vowpal Wabbit model and connect it to the left-hand input port. You can use a trained model created in the same experiment, or locate a saved model in the **Trained Models** group of Studio (classic)'s left navigation pane.

Restrictions

The model must be available in Azure Machine Learning Studio (classic); you cannot directly load a model from Azure storage.

Only Vowpal Wabbit 7-10 models are supported; you cannot connect saved models that were trained by using other algorithms, and you cannot use models that were trained using earlier or later versions.

3. In the **VW arguments** text box, type a set of valid command-line arguments to the Vowpal Wabbit executable.
For information about which Vowpal Wabbit arguments are supported and unsupported in Azure Machine Learning, see the [Technical Notes](#) section.
4. Click **Specify data type**, and select one of the supported data types from the list.

Scoring requires a single column of VW-compatible data.

If you have an existing file that was created in the SVMLight or VW formats, you can load it into the Azure ML workspace as a new dataset in one of these formats: Generic CSV without header, TSV without header.

The **VW** option requires that a label be present, but it is not used in scoring except for comparison.

5. Add the [Import Data](#) module and connect it to the right-hand input port of **Score Vowpal Wabbit Version 7-10**. Configure the [Import Data](#) to access the input data.

The input data for scoring must have been prepared ahead of time in one of the supported formats and stored in Azure blob storage.

6. Select the option, **Include an extra column containing labels**, if you want to output labels together with the scores.

Typically, when handling text data, Vowpal Wabbit does not require labels, and returns only the scores for each row of data.

7. Select the option, **Use cached results**, if you want to re-use results from a previous run, assuming the following conditions are met:

- A valid cache exists from a previous run.
- The input data and parameters settings of the module have not changed since the previous run.

Otherwise, the import process is repeated each time the experiment runs.

8. Run the experiment.

Results

After training is complete:

- To visualize the results, right-click the output of the [Score Vowpal Wabbit Version 7-10 Model](#) module .

The output indicates a prediction score normalized from 0 to 1.

Examples

For examples of how Vowpal Wabbit can be used in machine learning, see the [Azure AI Gallery](#):

- [Vowpal Wabbit sample](#)

This experiment demonstrates data preparation, training, and operationalization of a VW model.

The following video provides a walkthrough of the training and scoring process for Vowpal Wabbit:

<https://azure.microsoft.com/documentation/videos/text-analytics-and-vowpal-wabbit-in-azure-ml-studio/>

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Parameters

Vowpal Wabbit has many command-line options for choosing and tuning algorithms. A full discussion of these options is not possible here; we recommend that you view the [Vowpal Wabbit wiki page](#).

The following parameters are not supported in Azure Machine Learning Studio (classic).

- The input/output options specified in https://github.com/JohnLangford/vowpal_wabbit/wiki/Command-line-arguments

These properties are already configured automatically by the module.

- Additionally, any option that generates multiple outputs or takes multiple inputs is disallowed. These include `--cbt`, `--Lda`, and `--wap`.
- Only supervised learning algorithms are supported. This disallows these options: `-active`, `--rank`, `--search` etc.

All arguments other than those described above are allowed.

Expected inputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	Trained learner
Dataset	Data Table	Dataset to be scored

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
VW arguments	Any	String	none	Type Vowpal Wabbit arguments. The following arguments are not supported: - -i - -p or - -t
Include an extra column containing labels	True/False	Boolean	false	Specify whether the zipped file should include labels with the predictions
Specify data type	VW SVMLight	DataType	VW	Indicate whether the file format is SVMLight or Vowpal Wabbit

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with the prediction results

Exceptions

EXCEPTION	DESCRIPTION

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Text Analytics](#)

[Feature Hashing](#)

[Named Entity Recognition](#)

[Score Vowpal Wabbit 7-4 Model](#)

[Train Vowpal Wabbit 7-4 Model](#)

[Train Vowpal Wabbit 7-10 Model](#)

[A-Z Module List](#)

Score Vowpal Wabbit Version 8 Model

3/10/2021 • 4 minutes to read • [Edit Online](#)

Scores data using the Vowpal Wabbit machine learning system from the command line interface

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Score Vowpal Wabbit Version 8 Model** module in Azure Machine Learning Studio (classic), to generate scores for a set of input data, using an existing trained Vowpal Wabbit model.

This module provides the latest version of the Vowpal Wabbit framework, version 8. Use this module to score data using a trained model saved in the VW version 8 format.

If you have existing models created using an earlier version, use these modules:

- [Train Vowpal Wabbit 7-4 Model](#)
- [Score Vowpal Wabbit 7-4 Model](#)

How to configure Score Vowpal Wabbit Model 8

1. Add the **Score Vowpal Wabbit Version 8 Model** module to your experiment.
2. Add a trained Vowpal Wabbit model and connect it to the left-hand input port. You can use a trained model created in the same experiment, or locate a saved model in the **Trained Models** group of Studio (classic)'s left navigation pane. However, the model must be available in Azure Machine Learning Studio (classic); you cannot directly load a model from Azure storage.

NOTE

Only Vowpal Wabbit 8 models are supported; you cannot connect saved models that were trained by using other algorithms, and you cannot use models that were trained using earlier versions.

3. In the **VW arguments** text box, type a set of valid command-line arguments to the Vowpal Wabbit executable.

For information about which Vowpal Wabbit arguments are supported and unsupported in Azure Machine Learning, see the [Technical Notes](#) section.

4. Click **Specify data type**, and select one of the supported data types from the list.

Scoring requires a single column of VW-compatible data.

If you have an existing file that was created in the SVMLight or VW formats, you can load it into the Azure

ML workspace as a new dataset in one of these formats: Generic CSV without header, TSV without header.

The **VW** option requires that a label be present, but it is not used in scoring except for comparison.

5. Add an [Import Data](#) module and connect it to the right-hand input port of **Score Vowpal Wabbit**

Version 8. Configure the [Import Data](#) to access the input data.

The input data for scoring must have been prepared ahead of time in one of the supported formats and stored in Azure blob storage.

6. Select the option, **Include an extra column containing labels**, if you want to output labels together with the scores.

Typically, when handling text data, Vowpal Wabbit does not require labels, and will return only the scores for each row of data.

7. Select the option, **Include an extra column containing raw scores**, if you want to output raw scores together with the results.

TIP

This option is new for Vowpal Wabbit Version 8.

8. Select the option, **Use cached results**, if you want to re-use results from a previous run, assuming the following conditions are met:

- A valid cache exists from a previous run.
- The input data and parameters settings of the module have not changed since the previous run.

Otherwise, the import process is repeated each time the experiment runs.

9. Run the experiment.

Results

After training is complete:

- To visualize the results, right-click the output of the [Score Vowpal Wabbit Version 8 Model](#) module.

The output indicates a prediction score normalized from 0 to 1.

Examples

For examples of how Vowpal Wabbit can be used in machine learning, see the [Azure AI Gallery](#):

- [Vowpal Wabbit sample](#)

This experiment demonstrates data preparation, training, and operationalization of a VW model.

The following video provides a walkthrough of the training and scoring process for Vowpal Wabbit:

<https://azure.microsoft.com/documentation/videos/text-analytics-and-vowpal-wabbit-in-azure-ml-studio/>

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Parameters

Vowpal Wabbit has many command-line options for choosing and tuning algorithms. A full discussion of these options is not possible here; we recommend that you view the [Vowpal Wabbit wiki page](#).

The following parameters are not supported in Azure Machine Learning Studio (classic).

- The input/output options specified in https://github.com/JohnLangford/vowpal_wabbit/wiki/Command-line-arguments

These properties are already configured automatically by the module.

- Additionally, any option that generates multiple outputs or takes multiple inputs is disallowed. These include `--cbt`, `--Lda`, and `--wap`.
- Only supervised learning algorithms are supported. This disallows these options: `-active`, `--rank`, `--search` etc.

All arguments other than those described above are allowed.

Expected inputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	Trained learner
Dataset	Data Table	Dataset to be scored

Module Parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Specify data type	VW SVMLight	DataType	VW	Indicate whether the file type is SVMLight or Vowpal Wabbit
<i>VW arguments</i>	any	String	none	Type Vowpal Wabbit arguments. Do not include -i or -p, or -t
Include an extra column containing labels	True/False	Boolean	false	Specify whether the zipped file should include labels with the predictions
Include an extra column containing raw scores	True/False	Boolean	false	Specify whether the result should include an additional columns containing the raw scores (corresponding to --raw_predictions)

Outputs

NAME	TYPE	DESCRIPTION
Results dataset	Data Table	Dataset with the prediction results

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Text Analytics](#)

[Feature Hashing](#)

[Named Entity Recognition](#)

[Score Vowpal Wabbit 7-4 Model](#)

[Train Vowpal Wabbit 7-4 Model](#)

[Train Vowpal Wabbit 8 Model](#)

[A-Z Module List](#)

Train Vowpal Wabbit Version 7-4 Model

3/10/2021 • 10 minutes to read • [Edit Online](#)

Trains a model using version 7-4 of the Vowpal Wabbit machine learning system

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Train Vowpal Wabbit Version 7-4** module in Azure Machine Learning Studio (classic), to create a machine learning model by using an instance of Vowpal Wabbit (version 7-4).

To use Vowpal Wabbit for machine learning, format your input according to Vowpal Wabbit requirements, and save the data in an Azure blob. Use this module to specify Vowpal Wabbit command-line arguments.

When the experiment is run, an instance of Vowpal Wabbit is loaded into the experiment run-time, together with the specified data. When training is complete, the model is serialized back to the workspace. You can use the model immediately to score data. The trained model is also persisted in Azure storage so that you can use it later without having to reprocess the training data.

To incrementally train an existing model on new data, connect a saved model to the **Pre-trained model** input, and add the new data to the other input.

NOTE

Azure Machine Learning Studio (classic) hosts multiple versions of the Vowpal Wabbit framework. This module uses the 7-4 version of Vowpal Wabbit. If you create a model using this module, you must use the corresponding scoring module: [Score Vowpal Wabbit 7-4 Model](#).

For the latest version, use [Train Vowpal Wabbit Version 8 Model](#), together with its scoring module, [Score Vowpal Wabbit 8 Model](#).

What is Vowpal Wabbit?

Vowpal Wabbit (VW) is a fast, parallel machine learning framework that was developed for distributed computing by Yahoo! Research. Later it was ported to Windows and adapted by John Langford (Microsoft Research) for scientific computing in parallel architectures.

Features of Vowpal Wabbit that are important for machine learning include continuous learning (online learning), dimensionality reduction, and interactive learning. Vowpal Wabbit is also a solution for problems when you cannot fit the model data into memory.

The primary users of Vowpal Wabbit in Azure Machine Learning are data scientists who have previously used the framework for machine learning tasks such as classification, regression, topic modeling or matrix factorization. The Azure wrapper for Vowpal Wabbit has very similar performance characteristics to the on-

premise version, which means that users can continue to build models, retrain, and score using the powerful features and native performance of Vowpal Wabbit, while gaining the ability to easily publish the trained model as an operationalized service.

The [Feature Hashing](#) module also includes functionality provided by Vowpal Wabbit, that lets you transform text datasets into binary features using a hashing algorithm.

How to configure Vowpal Wabbit Version 8 Model

This section describes how to train a new model, and how to add new data to an existing model.

Unlike other modules in Studio (classic), this module both specifies the module parameters, and trains the model. If you have an existing model, you can add it as an optional input, to incrementally train the model.

- [Prepare input data in one of the required formats](#)
- [Train a new model](#)
- [Incrementally train an existing model](#)

Use of this module requires authentication to an Azure storage account.

Prepare the input data

To train a model using this module, the input dataset must consist of a single text column in one of the two supported formats: **LibSVM** or **VW**. This doesn't mean that Vowpal Wabbit analyzes only text data, only that the features and values must be prepared in the required text file format.

The data must be read from Azure storage. It is not possible to use [Export Data](#) to directly save the input file to Azure for use with Vowpal Wabbit, because the format requires some additional modification. You must ensure the data is in the correct format and then upload the data to Azure blob storage.

However, as a shortcut, you can use the [Convert to SVMLight](#) module to generate an SVMLight format file. Then, you can either upload the SVMLight format file to Azure blob storage and use it as the input, or you can modify the file slightly to conform to the Vowpal Wabbit input file requirements.

The Vowpal Wabbit data format has the advantage that it does not require a columnar format, which saves space when dealing with sparse data. For more information about this format, see the [Vowpal Wabbit wiki page](#).

Create and train a Vowpal Wabbit model

1. Add the [Train Vowpal Wabbit Version 7-4](#) module to your experiment.
2. Specify the account where the training data is stored. The trained model and hashing file are stored in the same location.
 - For **Azure storage account name**, type the name of the Azure storage account.
 - For **Azure storage key**, copy and paste the key that is provided for accessing the storage account,

If you don't have a key, see [How to regenerate storage access keys](#)

3. For **Azure container name**, type the name of a single container in the specified Azure storage account where the model training data is stored. Do not type the account name or any protocol prefix.

For example, if the full container path and name is `https://myaccount.blob.core.windows.net/vwmodels`, you should type just `vwmodels`. For more information about container names, see [Naming and Referencing Containers, Blobs, and Metadata](#).

4. In the **VW arguments** text box, type the command-line arguments for the Vowpal Wabbit executable.

For example, you might add `-L` to specify the learning rate, or `-b` to indicate the number of hashing

bits.

For more information, see the [parameters](#) section.

5. **Name of the input VW file:** Type the name of the file that contains the input data. The file must be an existing file in Azure blob storage, located in the previously specified storage account and container. The file must have been prepared using one of the supported formats.
6. **Name of the output readable model (--readable_model) file:** Type the name of a file where the trained model should be saved. The file must be saved within the same storage account and container as the input file.

This argument corresponds to the `--readable_model` parameter in the VW command line.

7. **Name of the output inverted hash (--invert_hash) file:** Type the name of the file in which the inverted hashing function should be saved. The file must be saved within the same storage account and container as the input file.

This argument corresponds to the `--invert_hash` parameter in the VW command line.

8. **Please specify file type:** Indicate which format your training data uses. Vowpal Wabbit supports these two input file formats:

- **VW** represents the internal format used by Vowpal Wabbit.
- **SVMLight** is a format used by some other machine learning tools.

9. Select the option, **Use cached results**, if you don't want to load the data from storage each time the experiment is re-run. Assuming no other parameters have changed and a valid cache can be found, Studio (classic) uses a cached version of the data.

If this option is deselected, the module always reads the data from storage.

10. Run the experiment.

11. After the model has been generated, right-click the output of **Train Vowpal Wabbit Version 7-4** and select **Save as trained model**, so that you can re-use and re-train the model later.

Retrain an existing Vowpal Wabbit model

Vowpal Wabbit supports incremental training by adding new data to an existing model. There are two ways to get an existing model for retraining:

- Use the output of another **Train Vowpal Wabbit Version 7-4** module in the same experiment.
- Locate a saved model in the **Trained Models** group of Studio (classic)'s left navigation pane, and drag it in to your experiment.

1. Add the **Train Vowpal Wabbit Version 7-4** module to your experiment.
2. Connect the previously trained model to the input port of **Train Vowpal Wabbit Version 7-4**.
3. In the **Properties** pane of **Train Vowpal Wabbit Version 7-4**, specify the location and format of the new training data.
4. Specify a name for the human-readable model output file, and another name for the hash file associated with the updated model.

NOTE

If there is an existing Vowpal Wabbit model or hash file in the specified location, the files are silently overwritten by the new trained model. To preserve intermediate models when retraining, you must change the storage location or make a local copy of the model files.

5. Run the experiment.
6. Right-click the module and select **Save as Trained Model** to preserve the updated model in your Azure Machine Learning workspace. If you don't specify a new name, the updated model overwrites the existing saved model.

Examples

For examples of how Vowpal Wabbit can be used in machine learning, see the [Azure AI Gallery](#):

- [Vowpal Wabbit sample](#)

This experiment demonstrates data preparation, training, and operationalization of a VW model.

Also, see these resources:

- Blog describing Vowpal Wabbit implementation and roadmap
<https://blogs.technet.com/b/machinelearning/archive/2014/10/02/vowpal-wabbit-modules-in-azureml.aspx>
- Video that demonstrates building and scoring a model using Vowpal Wabbit in Azure Machine Learning
<https://channel9.msdn.com/Blogs/Windows-Azure/Text-Analytics-and-Vowpal-Wabbit-in-Azure-ML-Studio>

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Advantages of Vowpal Wabbit

Vowpal Wabbit provides extremely fast learning over non-linear features like n-grams.

Vowpal Wabbit uses *online learning* techniques such as stochastic gradient descent (SGD) to fit a model one record at a time. Thus it iterates very quickly over raw data and can develop a good predictor faster than most other models. This approach also avoids having to read all training data into memory.

Vowpal Wabbit converts all data to hashes, not just text data but other categorical variables. Using hashes makes lookup of regression weights more efficient, which is critical for effective stochastic gradient descent.

During training, the module makes calls into a Vowpal Wabbit wrapper developed for Azure. The training data is downloaded in blocks from Azure, utilizing the high bandwidth between the worker roles executing the computations and the store, and is streamed to the VW learners. The resulting model is generally very compact due to the internal compression done by VW. The model is copied back to the experiment workspace where it can be utilized like other models in Azure Machine Learning.

Supported and unsupported parameters

You cannot use the following command-line arguments in Azure Machine Learning Studio (classic).

- The input/output options specified in https://github.com/JohnLangford/vowpal_wabbit/wiki/Command-line-arguments

These properties are already configured automatically by the module.

- Additionally, any option that generates multiple outputs or takes multiple inputs is disallowed. These include `--cbt`, `--Lda`, and `--wap`.
- Only supervised learning algorithms are supported. Therefore, these options are not supported: `-active`, `--rank`, `--search` etc.

All arguments other than those described above are allowed.

For a complete list of arguments, use the [Vowpal Wabbit wiki page](#).

Restrictions

Because the goal of the service is to support experienced users of Vowpal Wabbit, input data must be prepared ahead of time using the Vowpal Wabbit native text format, rather than the dataset format used by other modules.

Rather than using data in the Azure ML workspace, the training data is directly streamed from Azure, for maximal performance and minimal parsing overhead. For this reason, there is only limited interoperability between the VW modules and other modules in Azure ML.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Azure storage account name	any	String	none	Type the Azure storage account name
Azure storage key	any	SecureString	none	Provide the Azure storage key
Azure container name	any	String	none	Type the Azure container name
VW arguments	any	String	none	Specify any Vowpal Wabbit arguments. The argument <code>-fis</code> is not supported.
Name of the input VW file	any	String	none	Specify the name of an input file in the Vowpal Wabbit format
Name of the output readable model (<code>--readable_model</code>) file	any	String		If specified, outputs a readable model back to the Azure container. This argument is optional.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Name of the output inverted hash (--invert_hash) file	any	String		If specified, outputs a file containing the inverted hash function back to the Azure container. This argument is optional.
Please specify file type	VW SVMLight	DataType	VW	Indicate whether the file type uses the SVMLight format or the Vowpal Wabbit format.

Outputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	Trained learner

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

- [Text Analytics](#)
- [Feature Hashing](#)
- [Named Entity Recognition](#)
- [Score Vowpal Wabbit 7-4 Model](#)
- [Vowpal Wabbit Train](#)
- [A-Z Module List](#)

Train Vowpal Wabbit Version 7-10 Model

3/10/2021 • 10 minutes to read • [Edit Online](#)

Trains a model using version 7-10 of the Vowpal Wabbit machine learning system

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Train Vowpal Wabbit Version 7-10** module in Azure Machine Learning Studio (classic), to create a machine learning model using an instance of Vowpal Wabbit (version 7-10).

To use Vowpal Wabbit for machine learning, format your input according to Vowpal Wabbit requirements, and save the data in an Azure blob. Use this module to specify Vowpal Wabbit command-line arguments.

When the experiment is run, an instance of Vowpal Wabbit is loaded into the experiment run-time, together with the specified data. When training is complete, the model is serialized back to the workspace. You can use the model immediately to score data. The trained model is also persisted in Azure storage so that you can use it later without having to reprocess the training data.

To incrementally train an existing model on new data, connect a saved model to the **Pre-trained model** input, and add the new data to the other input.

NOTE

Azure Machine Learning Studio (classic) hosts multiple versions of the Vowpal Wabbit framework. This module uses the version 7-10 of the Vowpal Wabbit framework.

If you need to build or score a model based on a previous version (7-4 or 7-6), use these modules: [Train Vowpal Wabbit 7-4 Model](#) and [Score Vowpal Wabbit 7-4 Model](#).

For the latest version, use [Train Vowpal Wabbit Version 8 Model](#), together with its scoring module, [Score Vowpal Wabbit 8 Model](#).

What is Vowpal Wabbit?

Vowpal Wabbit (VW) is a fast, parallel machine learning framework that was developed for distributed computing by Yahoo! Research. Later it was ported to Windows and adapted by John Langford (Microsoft Research) for scientific computing in parallel architectures.

Features of Vowpal Wabbit that are important for machine learning include continuous learning (online learning), dimensionality reduction, and interactive learning. Vowpal Wabbit is also a solution for problems when you cannot fit the model data into memory.

The primary users of Vowpal Wabbit in Azure Machine Learning are data scientists who have previously used the framework for machine learning tasks such as classification, regression, topic modeling or matrix

factorization. The Azure wrapper for Vowpal Wabbit has very similar performance characteristics to the on-premise version, which means that users can continue to build models, retrain, and score using the powerful features and native performance of Vowpal Wabbit, while gaining the ability to easily publish the trained model as an operationalized service.

The [Feature Hashing](#) module also includes functionality provided by Vowpal Wabbit, that lets you transform text datasets into binary features using a hashing algorithm.

How to configure Vowpal Wabbit Version 7-10 Model

This section describes how to train a new model, and how to add new data to an existing model.

Unlike other modules in Studio (classic), this module both specifies the module parameters, and trains the model. If you have an existing model, you can add it as an optional input, to incrementally train the model.

- [Prepare input data in one of the required formats](#)
- [Train a new model](#)
- [Incrementally train an existing model](#)

Use of this module requires authentication to an Azure storage account.

Prepare the input data

To train a model using this module, the input dataset must consist of a single text column in one of the two supported formats: **LibSVM** or **VW**.

This doesn't mean that Vowpal Wabbit analyzes only text data, just that the features and values must be prepared in the required text file format.

The data must be read from Azure storage. It is not possible to use [Export Data](#) to directly save the input file to Azure for use with Vowpal Wabbit, because the format requires some additional modification. You must ensure the data is in the correct format and then upload the data to Azure blob storage.

However, as a shortcut, you can use the [Convert to SVMLight](#) module to generate an SVMLight format file. Then, you can either upload the SVMLight format file to Azure blob storage and use it as the input, or you can modify the file slightly to conform to the Vowpal Wabbit input file requirements.

The Vowpal Wabbit data format has the advantage that it does not require a columnar format, which saves space when dealing with sparse data. For more information about this format, see the [Vowpal Wabbit wiki page](#).

Create and train a Vowpal Wabbit model

1. Add the [Train Vowpal Wabbit Version 7-10](#) module to your experiment.
2. Specify the account where the training data is stored. The trained model and hashing file are stored in the same location.
 - For **Azure storage account name**, type the name of the Azure storage account.
 - For **Azure storage key**, copy and paste the key that is provided for accessing the storage account,

If you don't have a key, see [How to regenerate storage access keys](#)

3. For **Azure container name**, type the name of a single container in the specified Azure storage account where the model training data is stored. Do not type the account name or any protocol prefix.

For example, if the full container path and name is `https://myaccount.blob.core.windows.net/vwmodels`, you should type just `vwmodels`. For more information about container names, see [Naming and Referencing Containers, Blobs, and Metadata](#).

4. In the **VW arguments** text box, type the command-line arguments for the Vowpal Wabbit executable.

For example, you might add `-l` to specify the learning rate, or `-b` to indicate the number of hashing bits.

For more information, see the [parameters](#) section.

5. **Name of the input VW file:** Type the name of the file that contains the input data. The file must be an existing file in Azure blob storage, located in the previously specified storage account and container. The file must have been prepared using one of the supported formats.

6. **Name of the output readable model (--readable_model) file:** Type the name of a file where the trained model should be saved. The file must be saved within the same storage account and container as the input file.

This argument corresponds to the `--readable_model` parameter in the VW command line.

7. **Name of the output inverted hash (--invert_hash) file:** Type the name of the file in which the inverted hashing function should be saved. The file must be saved within the same storage account and container as the input file.

This argument corresponds to the `--invert_hash` parameter in the VW command line.

8. **Please specify file type:** Indicate which format your training data uses. Vowpal Wabbit supports these two input file formats:

- **VW** represents the internal format used by Vowpal Wabbit . See the [Vowpal Wabbit wiki page](#) for details.
- **SVMLight** is a format used by some other machine learning tools.

9. Select the option, **Use cached results**, if you don't want to load the data from storage each time the experiment is re-run. Assuming no other parameters have changed and a valid cache can be found, Studio (classic) uses a cached version of the data.

If this option is deselected, the module always reads the data from storage.

10. Run the experiment.

11. After the model has been generated, right-click the output and select **Save as trained model**, so that you can re-use and re-train the model later.

Retrain an existing Vowpal Wabbit model

Vowpal Wabbit supports incremental training by adding new data to an existing model. There are two ways to get an existing model for retraining:

- Use the output of another **Train Vowpal Wabbit Version 8** module in the same experiment.
 - Locate a saved model in the **Trained Models** group in Studio (classic), and drag it in to your experiment.
1. Add the **Train Vowpal Wabbit Version 8** module to your experiment.
 2. Connect the previously trained model to the input port of **Train Vowpal Wabbit Version 8**.
 3. In the **Properties** pane of **Train Vowpal Wabbit Version 8**, specify the location and format of the new training data.
 4. Specify a name for the human-readable model output file, and another name for the hash file associated with the updated model.

NOTE

If there is an existing Vowpal Wabbit model or hash file in the specified location, the files are silently overwritten by the new trained model. To preserve intermediate models when retraining, you must change the storage location or make a local copy of the model files.

5. Run the experiment.
6. Right-click the module and select **Save as Trained Model** to preserve the updated model in your Azure Machine Learning workspace. If you don't specify a new name, the updated model overwrites the existing saved model.

Examples

For examples of how Vowpal Wabbit can be used in machine learning, see the [Azure AI Gallery](#):

- [Vowpal Wabbit sample](#)

This experiment demonstrates data preparation, training, and operationalization of a VW model.

Also, see these resources:

- Blog describing Vowpal Wabbit implementation and roadmap
[Vowpal Wabbit Modules in AzureML](#)
- Video that demonstrates building and scoring a model using Vowpal Wabbit in Azure Machine Learning
[Text Analytics and Vowpal Wabbit in Azure Machine Learning Studio \(classic\)](#)

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Advantages of Vowpal Wabbit

Vowpal Wabbit provides extremely fast learning over non-linear features like n-grams.

Vowpal Wabbit uses *online learning* techniques such as stochastic gradient descent (SGD) to fit a model one record at a time. Thus it iterates very quickly over raw data and can develop a good predictor faster than most other models. This approach also avoids having to read all training data into memory.

Vowpal Wabbit converts all data to hashes, not just text data but other categorical variables. Using hashes makes lookup of regression weights more efficient, which is critical for effective stochastic gradient descent.

During training, the module makes calls into a Vowpal Wabbit wrapper developed for Azure. The training data is downloaded in blocks from Azure, utilizing the high bandwidth between the store and the worker roles executing the computations, and is streamed to the VW learners. The resulting model is generally very compact due to the internal compression done by VW. The model is copied back to the experiment workspace where it can be utilized like other models in Azure Machine Learning.

Supported and unsupported parameters

This section describes support for Vowpal Wabbit command line parameters in Azure Machine Learning Studio (classic).

You cannot use the following command-line arguments in Azure Machine Learning Studio (classic).

- The input/output options specified in [Vowpal Wabbit Wiki - Command-line-arguments](#)

These properties are configured automatically by the module.

- Any option that generates multiple outputs or takes multiple inputs is disallowed. These include:

`--cbt` , `--l1da` , `--wap`

- Only supervised learning algorithms are supported. Therefore, options such as these are not supported:

`-active` , `--rank` , `--search`

All arguments other than those described above are allowed.

For a complete list of arguments, use the [Vowpal Wabbit wiki page](#).

Restrictions

Because the goal of the service is to support experienced users of Vowpal Wabbit, input data must be prepared ahead of time using the Vowpal Wabbit native text format, rather than the dataset format used by other modules.

Rather than using data in the Azure ML workspace, the training data is directly streamed from Azure, for maximal performance and minimal parsing overhead. For this reason, there is only limited interoperability between the VW modules and other modules in Azure ML.

Module parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Azure storage account name	any	String		Type the Azure storage account name
Azure storage key	any	SecureString		Provide the Azure storage key
Azure container name	any	String		Type the Azure container name
VW arguments	any	String		Specify any Vowpal Wabbit arguments. The argument <code>-fis</code> is not supported.
Name of the input VW file	any	String		Specify the name of an input file in the Vowpal Wabbit format
Name of the output readable model (<code>--readable_model</code>) file	any	String		If specified, outputs a readable model back to the Azure container. This argument is optional.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Name of the output inverted hash (--invert_hash) file	any	String		If specified, outputs a file containing the inverted hash function back to the Azure container. This argument is optional.
Please specify file type	VW SVMLight	DataType	VW	Indicate whether the file type uses the SVMLight format or the Vowpal Wabbit format.

Outputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	Trained learner

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

[Text Analytics](#)

[Feature Hashing](#)

[Named Entity Recognition](#)

[Score Vowpal Wabbit 7-4 Model](#)

[Score Vowpal Wabbit 7-10 Model](#)

[Train Vowpal Wabbit 7-4 Model](#)

[A-Z Module List](#)

Train Vowpal Wabbit Version 8 Model

3/10/2021 • 10 minutes to read • [Edit Online](#)

Trains a model using version 8 of the Vowpal Wabbit machine learning system

Category: [Text Analytics](#)

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Train Vowpal Wabbit Version 8** module in Azure Machine Learning Studio (classic), to create a machine learning model by using Vowpal Wabbit (version 8).

To use Vowpal Wabbit for machine learning, format your input according to Vowpal Wabbit requirements, and save the data in an Azure blob. Use this module to specify Vowpal Wabbit command-line arguments.

When the experiment is run, an instance of Vowpal Wabbit is loaded into the experiment run-time, together with the specified data. When training is complete, the model is serialized back to the workspace. You can use the model immediately to score data. The trained model is also persisted in Azure storage so that you can use it later without having to reprocess the training data.

To incrementally train an existing model on new data, connect a saved model to the **Pre-trained model** input, and add the new data to the other input.

NOTE

Azure Machine Learning Studio (classic) hosts multiple versions of the Vowpal Wabbit framework. This module uses the latest version of the Vowpal Wabbit framework, which is version 8. To score new input data, you must use [Score Vowpal Wabbit Version 8 Model](#).

Vowpal Wabbit versions 7-4 or 7-6: [Train Vowpal Wabbit 7-4 Model](#) and [Score Vowpal Wabbit 7-4 Model](#).

Vowpal Wabbit version 7-10: [Train Vowpal Wabbit 7-10 Model](#) and [Score Vowpal Wabbit 7-10 Model](#).

What is Vowpal Wabbit?

Vowpal Wabbit (VW) is a fast, parallel machine learning framework that was developed for distributed computing by Yahoo! Research. Later it was ported to Windows and adapted by John Langford (Microsoft Research) for scientific computing in parallel architectures.

Features of Vowpal Wabbit that are important for machine learning include continuous learning (online learning), dimensionality reduction, and interactive learning. Vowpal Wabbit is also a solution for problems when you cannot fit the model data into memory.

The primary users of Vowpal Wabbit are data scientists who have previously used the framework for machine learning tasks such as classification, regression, topic modeling or matrix factorization. The Azure wrapper for Vowpal Wabbit has very similar performance characteristics to the on-premises version, so you can use the

powerful features and native performance of Vowpal Wabbit, and easily publish the trained model as an operationalized service.

The [Feature Hashing](#) module also includes functionality provided by Vowpal Wabbit, that lets you transform text datasets into binary features using a hashing algorithm.

How to configure Vowpal Wabbit Version 8 Model

This section describes how to train a new model, and how to add new data to an existing model.

Unlike other modules in Studio (classic), this module both specifies the module parameters, and trains the model. If you have an existing model, you can add it as an optional input, to incrementally train the model.

- [Prepare input data in one of the required formats](#)
- [Train a new model](#)
- [Incrementally train an existing model](#)

Use of this module requires authentication to an Azure storage account.

Prepare the input data

To train a model using this module, the input dataset must consist of a single text column in one of the two supported formats: **LibSVM** or **VW**. This doesn't mean that Vowpal Wabbit analyzes only text data, only that the features and values must be prepared in the required text file format.

The data must be read from Azure storage. It is not possible to use [Export Data](#) to directly save the input file to Azure for use with Vowpal Wabbit, because the format requires some additional modification. You must ensure the data is in the correct format and then upload the data to Azure blob storage.

However, as a shortcut, you can use the [Convert to SVMLight](#) module to generate an SVMLight format file. Then, you can either upload the SVMLight format file to Azure blob storage and use it as the input, or you can modify the file slightly to conform to the Vowpal Wabbit input file requirements.

The Vowpal Wabbit data format has the advantage that it does not require a columnar format, which saves space when dealing with sparse data. For more information about this format, see the [Vowpal Wabbit wiki page](#).

Create and train a Vowpal Wabbit model

1. Add the [Train Vowpal Wabbit Version 8](#) module to your experiment.
2. Specify the account where the training data is stored. The trained model and hashing file are stored in the same location.
 - For **Azure storage account name**, type the name of the Azure storage account.
 - For **Azure storage key**, copy and paste the key that is provided for accessing the storage account,

If you don't have a key, see [How to regenerate storage access keys](#)

3. For **Azure container name**, type the name of a single container in the specified Azure storage account where the model training data is stored. Do not type the account name or any protocol prefix.

For example, if the full container path and name is `https://myaccount.blob.core.windows.net/vwmodels`, you should type just `vwmodels`. For more information about container names, see [Naming and Referencing Containers, Blobs, and Metadata](#).

4. In the **VW arguments** text box, type the command-line arguments for the Vowpal Wabbit executable.

For example, you might add `-l` to specify the learning rate, or `-b` to indicate the number of hashing bits.

For more information, see the [Vowpal Wabbit parameters](#) section.

5. **Name of the input VW file:** Type the name of the file that contains the input data. The file must be an existing file in Azure blob storage, located in the previously specified storage account and container. The file must have been prepared using one of the supported formats.
6. **Name of the output readable model (--readable_model) file:** Type the name of a file where the trained model should be saved. The file must be saved within the same storage account and container as the input file.

This argument corresponds to the `--readable_model` parameter in the VW command line.

7. **Name of the output inverted hash (--invert_hash) file:** Type the name of the file in which the inverted hashing function should be saved. The file must be saved within the same storage account and container as the input file.

This argument corresponds to the `--invert_hash` parameter in the VW command line.

8. **Please specify file type:** Indicate which format your training data uses. Vowpal Wabbit supports these two input file formats:

- **VW** represents the internal format used by Vowpal Wabbit . See the [Vowpal Wabbit wiki page](#) for details.
- **SVMLight** is a format used by some other machine learning tools.

9. Select the option, **Use cached results**, if you don't want to load the data from storage each time the experiment is re-run. Assuming no other parameters have changed and a valid cache can be found, Studio (classic) uses a cached version of the data.

If this option is deselected, the module always reads the data from storage.

10. Run the experiment.

11. When training is complete, right-click the output and select **Save as Trained Model** to save the model to your Studio (classic) workspace.

Retrain an existing Vowpal Wabbit model

Vowpal Wabbit supports incremental training by adding new data to an existing model. There are two ways to get an existing model for retraining:

- Use the output of another **Train Vowpal Wabbit Version 8** module in the same experiment.
- Locate a saved model in the **Trained Models** group of Studio (classic)'s left navigation pane, and drag it in to your experiment.

1. Add the **Train Vowpal Wabbit Version 8** module to your experiment.
2. Connect the previously trained model to the input port of **Train Vowpal Wabbit Version 8**.
3. In the **Properties** pane of **Train Vowpal Wabbit Version 8**, specify the location and format of the new training data.
4. Specify a name for the human-readable model output file, and another name for the hash file associated with the updated model.

NOTE

If there is an existing Vowpal Wabbit model or hash file in the specified location, the files are silently overwritten by the new trained model. To preserve intermediate models when retraining, you must change the storage location or make a local copy of the model files.

5. Run the experiment.
6. Right-click the module and select **Save as Trained Model** to preserve the updated model in your Azure Machine Learning workspace. If you don't specify a new name, the updated model overwrites the existing saved model.

Examples

For examples of how Vowpal Wabbit can be used in machine learning, see the [Azure AI Gallery](#):

- [Vowpal Wabbit sample](#)

This experiment demonstrates data preparation, training, and operationalization of a VW model.

Also, see these resources:

- Blog describing Vowpal Wabbit implementation and roadmap
<https://blogs.technet.com/b/machinelearning/archive/2014/10/02/vowpal-wabbit-modules-in-azureml.aspx>
- Video that demonstrates building and scoring a model using Vowpal Wabbit in Azure Machine Learning
<https://channel9.msdn.com/Blogs/Windows-Azure/Text-Analytics-and-Vowpal-Wabbit-in-Azure-ML-Studio>

Technical notes

This section contains implementation details, tips, and answers to frequently asked questions.

Advantages of Vowpal Wabbit

Vowpal Wabbit provides extremely fast learning over non-linear features like n-grams.

Vowpal Wabbit uses *online learning* techniques such as stochastic gradient descent (SGD) to fit a model one record at a time. Thus it iterates very quickly over raw data and can develop a good predictor faster than most other models. This approach also avoids having to read all training data into memory.

Vowpal Wabbit converts all data to hashes, not just text data but other categorical variables. Using hashes makes lookup of regression weights more efficient, which is critical for effective stochastic gradient descent.

During training, the module makes calls into a Vowpal Wabbit wrapper developed for Azure. The training data is downloaded in blocks from Azure, utilizing the high bandwidth between the worker roles executing the computations and the store, and is streamed to the VW learners. The resulting model is generally very compact due to the internal compression done by VW. The model is copied back to the experiment workspace where it can be utilized like other models in Azure Machine Learning.

Supported and unsupported parameters

This section describes support for Vowpal Wabbit command line parameters in Azure Machine Learning Studio (classic).

Generally, all but a limited set of arguments are supported. For a complete list of arguments, use the [Vowpal Wabbit wiki page](#).

The following parameters are not supported:

- The input/output options specified in https://github.com/JohnLangford/vowpal_wabbit/wiki/Command-line-arguments

These properties are already configured automatically by the module.

- Additionally, any option that generates multiple outputs or takes multiple inputs is disallowed. These include `--cbt`, `--Lda`, and `--wap`.
- Only supervised learning algorithms are supported. Therefore, these options are not supported: `-active`, `--rank`, `--search` etc. **### Restrictions**

Restrictions

Because the goal of the service is to support experienced users of Vowpal Wabbit, input data must be prepared ahead of time using the Vowpal Wabbit native text format, rather than the dataset format used by other modules.

Rather than using data in the Azure ML workspace, the training data is directly streamed from Azure, for maximal performance and minimal parsing overhead. For this reason, there is only limited interoperability between the VW modules and other modules in Azure ML.

Module parameters

NAME	RANGE	TYPE	OPTIONAL	DEFAULT	DESCRIPTION
Please specify file type	VW SVMLight	DataType	Required	VW	Indicate whether the file type is SVMLight or Vowpal Wabbit.
Azure storage account name	any	String	Required		Type the Azure storage account name
Azure storage key	any	SecureString	Required		Provide the Azure storage key
Azure container name	any	String	Required		Type the Azure container name
VW arguments	any	String	Optional		Specify any Vowpal Wabbit arguments. Do not include -f.
Name of the input VW file	any	String	Required		Specify the name of an input file in the Vowpal Wabbit format
Name of the output readable model (--readable_model) file	any	String	Optional		If specified, outputs a readable model back to the Azure container.

NAME	RANGE	TYPE	OPTIONAL	DEFAULT	DESCRIPTION
Name of the output inverted hash (--invert_hash) file	String	String	Optional		If specified, outputs a file containing the inverted hash function back to the Azure container.

Outputs

NAME	TYPE	DESCRIPTION
Trained model	ILearner interface	Trained learner

Exceptions

EXCEPTION	DESCRIPTION
Error 0001	Exception occurs if one or more specified columns of data set couldn't be found.
Error 0003	Exception occurs if one or more of inputs are null or empty.
Error 0004	Exception occurs if parameter is less than or equal to specific value.
Error 0017	Exception occurs if one or more specified columns have type unsupported by current module.

For a list of errors specific to Studio (classic) modules, see [Machine Learning Error codes](#).

For a list of API exceptions, see [Machine Learning REST API Error Codes](#).

See also

- [Text Analytics](#)
- [Feature Hashing](#)
- [Named Entity Recognition](#)
- [Score Vowpal Wabbit 7-4 Model](#)
- [Score Vowpal Wabbit Version 8 Model](#)
- [Train Vowpal Wabbit 7-4 Model](#)
- [A-Z Module List](#)

Time Series

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the modules in Azure Machine Learning Studio (classic) that are specifically designed for working with time series data.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

If you need help determining whether you need an algorithm specially for time series, or another type of algorithm, see these resources:

- [Machine learning algorithm cheat sheet for Microsoft Azure Machine Learning Studio \(classic\)](#)
- [How to choose Azure Machine Learning algorithms for clustering, classification, or regression](#)

Time series modules

Azure Machine Learning Studio (classic) provides the following module for analyzing time series:

- [Time Series Anomaly Detection](#)

Additionally, you can use the following modules to perform custom time series analyses, by using the R or Python languages.

- [Execute R Script](#)
- [Execute Python Script](#)

See also

- [Classification models](#)
- [Regression models](#)

Time Series Anomaly Detection

3/10/2021 • 9 minutes to read • [Edit Online](#)

Detects anomalies in the input time series data.

Category: Time Series

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

Module overview

This article describes how to use the **Time Series Anomaly Detection** module in Azure Machine Learning Studio (classic), to detect anomalies in time series data. The module learns the normal operating characteristics of a time series that you provide as input, and uses that information to detect deviations from the normal pattern. The module can detect both changes in the overall trend, and changes in the magnitude or range of values.

Detecting changes in time series data has wide applications. For example, you could use it for near-real-time monitoring of sensors, networks, or resource usage. By tracking service errors, service usage, and other KPIs, you can respond quickly to critical anomalies. Other applications include health care and finance.

Anomaly detection methods

Anomaly detection is the problem of finding patterns in data that do not conform to a model of "normal" behavior. Typical approaches for detecting such changes either use simple human computed thresholds, or mean and standard deviation to determine when data deviates significantly from the mean.

However, such simple approaches are not easily adapted to time series data:

- Large numbers of false anomalies are generated
- The methods are not applicable to changing data values
- Cannot easily scale to large time series

This module provides two additional methods for assessing variance from a time series trend:

- Measuring the magnitude of upward and downward changes

For example, the number of requests on a web service might be stable for some time and then dramatically increase.

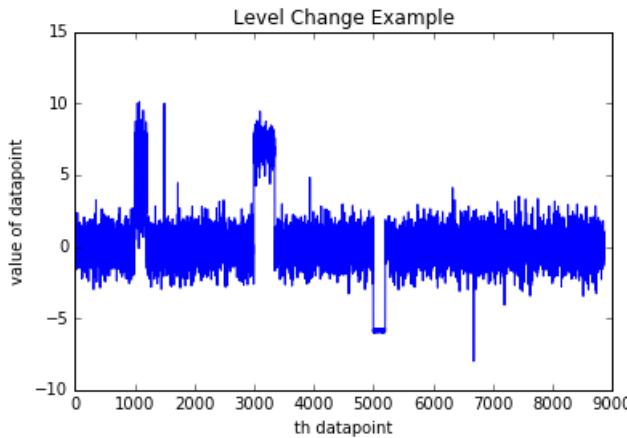
- Measuring the direction and duration of trends: positive vs. negative changes

For example, a persistent upward trend in the length of a service queue might indicate an underlying issue. Even though the overall trend is consistently increasing and therefore might be considered stable, a change in slope could be flagged as an anomaly. Conversely, if you are monitoring memory usage of a server, a constant decrease in free memory size might indicate an issue.

Examples of anomalous patterns in time series

Upward and downward level changes

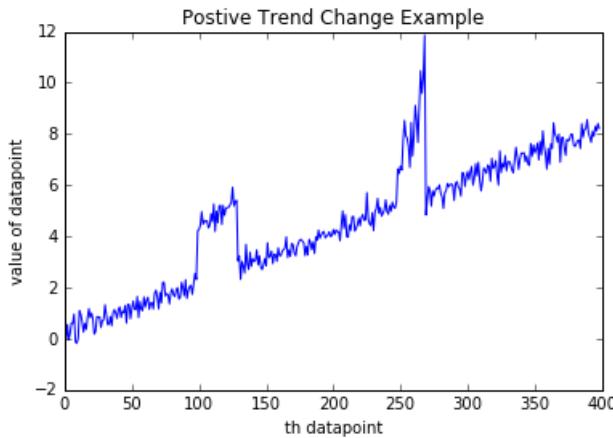
For example, assume you have been monitoring the number of requests per day to a web service over a period of time, and the number of requests appears to stay within a certain range. However, after an update to the web service, the number of requests to that web service changes. The new trend might be either higher or lower than the original trend; both upward and downward spikes can be detected.



Positive or negative trends changes

For example, assume you are monitoring the length of a queue on a service support site. A persistent upward trend might indicate an underlying service issue.

In other cases, a persistent negative trend might be the anomaly. For example, if you are monitoring memory usage on a server, when free memory size shrinks, it could indicate a potential memory leak.



Resources

For more information about the research underlying this approach, see these articles:

- Shen-Shyang Ho; Wechsler, H., "A Martingale Framework for Detecting Changes in Data Streams by Testing Exchangeability," Pattern Analysis and Machine Intelligence, IEEE Transactions , vol.32, no.12, pp.2113,2127, Dec. 2010

[Sources and citations \(Microsoft Academic\)](#)

- Valentina Fedorova, Alex J. Gammerman, Ilia Nouretdinov, Vladimir Vovk, "Plug-in martingales for testing exchangeability on-line", ICML 2012

[Sources and citations \(Microsoft Academic\)](#)

- Vladimir Vovk, Ilia Nouretdinov, Alex J. Gammerman, "Testing Exchangeability Online", ICML 2003.

How to configure Time Series Anomaly Detection

1. Add the **Time Series Anomaly Detection** module to your experiment and connect the dataset that contains the time series.

The dataset used as input must contain at least one column containing datetime values in string format, and another column that contains the trend values, in a numeric format. Other columns are ignored.

Because each row corresponds to a data point in the time series representing the value at that time, the time values should be unique.

2. **Data Column:** Select a single column in your dataset that contains numeric data values. These values are the data points in the trend that you want to model, such as population totals over time, costs per month, or temperatures over some period.
3. **Time Column:** Select a single column in your dataset that contains the associated time series value.

The column must contain valid datetime values, meaning that all dates must be within the range of dates supported by the .NET Framework.

The Time column must use the `DateTime` data type. If your dates are in string format, you can cast them using the [Apply SQL Transformation](#) module, or convert them using the [Execute R Script](#) module. If your dates are represented as integers, you must also use an appropriate datetime conversion function to represent the values using a valid datetime format.

For example, the following SQL statement changes an Excel serial date value to a datetime format:

```
SELECT CAST([SerialDate] AS SmallDateTime) as [NewValidDate] from t1;
```

After the date values are in the correct format, use the [Edit Metadata](#) module to set the column type to `DateTime`.

4. **Martingale Type:** Select the martingale function to use.

- **PowerAvg.** This option is a marginalized implementation of the power martingale.

The default value is **PowerAvg** with no additional parameters. This option gives a more stable anomaly detector and should be suitable for most needs.

- **Power.** A non-marginalized implementation of the power martingale.

The **Power** option provide users with the option to provide a value between 0 and 1 for the **Epsilon** parameter to control the sensitivity of the detector. Generally, a higher epsilon value means higher sensitivity to anomalies but less certainty.

For a definition of martingales and the methods used in this module, see: [Anomaly detection using machine learning to detect abnormalities in time series data](#)

5. **Strangeness Function Type:** This option is used to specific different types of anomalies. Three options are supported, which require no further parameters:

- **RangePercentile.**

This is the default, and is mostly used for detecting level changes.

- **SlowPosTrend.** Choose this option to detect positive trend changes.

- **SlowNegTrend.** Choose this option to detect negative trend changes.

6. Length of Martingale and Strangeness Values: Specify the size of the history window, which is used to compute martingale values over the look-back history.

The default value is 500, but you can specify any integer between 0 and 5000. For large time series, the default value should work well. For smaller time series, you can try to estimate the value for the expected length of the abnormal behavior.

We recommend that you generally set these two parameters to the same value.

7. Length of Strangeness Values: Specify the length of the history window used to compute strangeness at each data point.

The default value is 500, but you can specify any integer between 0 and 5000.

This parameter has the same restrictions as **Length of Martingale**. That is, you should set the value to the estimated number of data points that are needed to learn "normal" behavior.

The default of 500 works well for most cases, but if the scale over which "normalcy" is measured varies, it might be beneficial to make **Length of Strangeness Values** a greater value than **Length of Martingale**.

For example, if you are monitoring errors, and assume that data points are captured at 15-min intervals, the time needed to learn the normal trend might vary greatly from month to month.

In general, using a larger window size leads to slower performance, because the module has to learn over a larger dataset.

We recommend that you generally set these two parameters to the same value.

8. Alert Threshold: Specify a value above which the anomaly score generates an alert.

The default value is 3.25, meaning that an alert is generated for every row containing a score of 3.25 or more. Y

You can specify any floating point number between 0 and 100. However, there is a tradeoff between sensitivity and confidence in the choice of threshold:

- A lower threshold would make the detector more sensitive to anomalies and generate more alerts.
- A lower threshold might result in normal changes being misclassified as anomalies.

9. Run the experiment.

Note that you do not need to train this model separately; the algorithm learns the pattern from the data you provide as input to this module.

Results

When training is complete, the module outputs a time series that is the same length as the input time series; however, two columns are added to indicate values that are potentially anomalous.

- **Anomaly Score:** The first column contains a score that represents the likelihood that the time series value is anomalous.
- **Alert:** This column contain a flag with a value of 0 or 1, where 1 means that an anomaly was detected. You can set the threshold for generating the alert based on the score column, but setting the **Alert Threshold** parameter.

Examples

The following examples demonstrate how to set the martingale function to detect anomalies and how to

interpret the results.

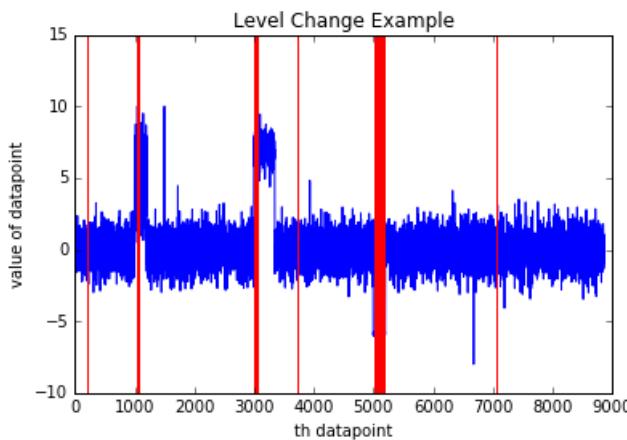
Detect level changes

To illustrate the impact of different settings, the example dataset used in this example consists of 8870 data points, with three level changes. Based on this data, we created two models, using the following parameters.

- **Martingale Type:** PowerAvg
- **Strangeness function type:** RangePercentile
- **Length of Martingale** = 50
- **Length of Strangeness Values** = 50

The model was trained on the incoming data, but a different value was applied for **Alert Threshold**. The results of prediction are plotted in the following images for Model 1 and Model 2. In these graphs, the blue lines represent the data values, and the red lines represent the **alerts** raised for an anomaly.

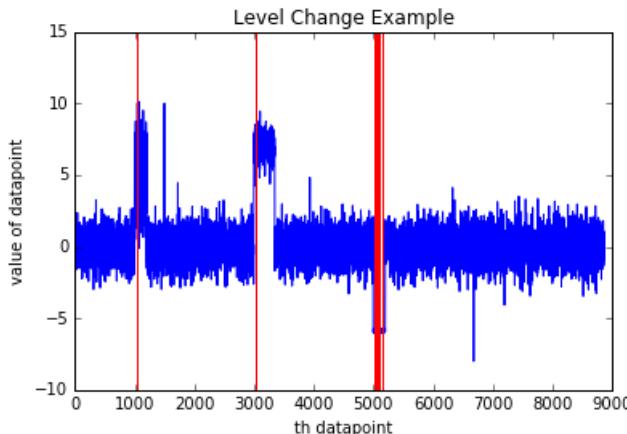
Alert threshold of 0.9



In this model, the threshold value is lower, and therefore alerts are raised (anomalies detected) even when the changes are momentary.

Depending on the type of time series you are monitoring, some of these alerts might be considered false alerts. However, a lower threshold might be preferable if you cannot afford to overlook any anomaly.

Alert threshold of 3.25



In this model, the alert threshold was much higher, and as a result, the model detects only those changes that persist longer. A higher threshold for alerting might be more desirable in a scenario where you only want to catch more long-lasting changes.

Detect positive slope changes

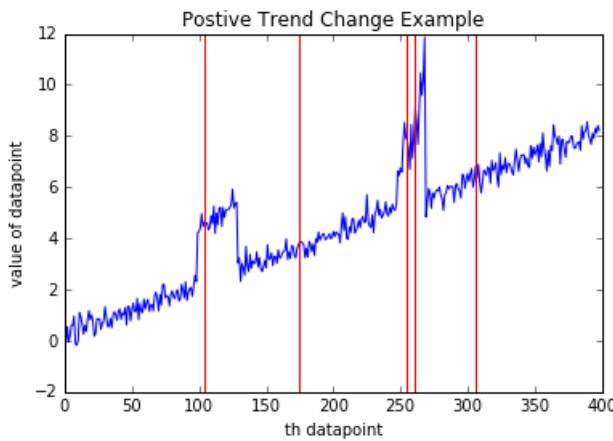
To illustrate this option for anomaly detection, we used an example dataset containing 300 data points. All the points formed a positive trend overall, with two anomalies.

Again, we created two models using parameters that were identical except for the alert threshold value.

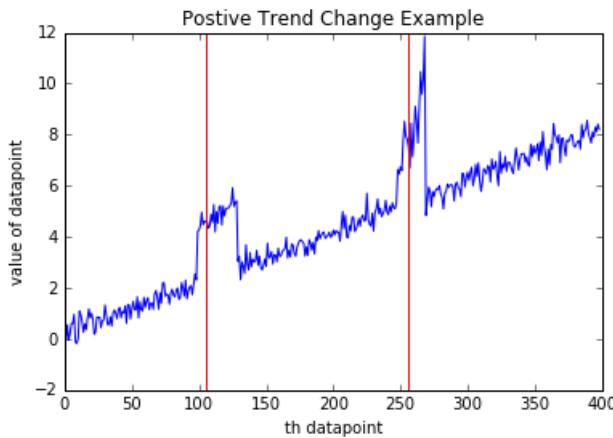
- Martingale Type: PowerAvg
- Strangeness function type: SlowPosTrend
- Length of Martingale = 50
- Length of Strangeness Values = 50

In these graphs, the blue lines represent data values and the red lines represent alerts raised for an anomaly.

Alert Threshold of 4.25



Alert Threshold of 6.0



We recommend that you experiment with different alert threshold values to find the appropriate level of sensitivity for your anomaly detection scenario.

Expected inputs

NAME	TYPE	DESCRIPTION
Input data containing time stamps and values	Data Table	Input data containing date-time stamps and values.

Module parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Value Column	ColumnSelection		Required		Choose the column that contains the time series to track
Window Size	Integer		Required		Specify a value that controls the size of the analysis window
Threshold	Float		Optional		Specify a value that determines the threshold on the score to identify an anomaly

Outputs

NAME	TYPE	DESCRIPTION
Time series annotated with anomaly scores	Data Table	Dataset with anomaly intervals.

See also

- [Anomaly Detection](#)
- [One-Class Support Vector Machine](#)
- [PCA-Based Anomaly Detection](#)

Machine Learning Module Data Types

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes the .NET data types that are supported in Microsoft Azure Machine Learning Studio (classic) for external data. It also describes the custom data type classes that are used for passing data between modules within an experiment.

Table of .NET data types

The following .NET types are supported by Machine Learning Studio (classic) modules.

.NET DATA TYPE	COMMENTS
Boolean	https://msdn.microsoft.com/library/wts33hb3.aspx
Int16	https://msdn.microsoft.com/library/system.int16(v=vs.110).aspx
Int32	https://msdn.microsoft.com/library/06bkb8w2.aspx
Int64	https://msdn.microsoft.com/library/system.int64.aspx
Single	https://msdn.microsoft.com/library/system.single(v=vs.110).aspx
Double	https://msdn.microsoft.com/library/system.double(v=vs.110).aspx
String	https://msdn.microsoft.com/library/system.string(v=vs.110).aspx
datetime	https://msdn.microsoft.com/library/system.datetime(v=vs.110).aspx
DateTimeOffset	https://msdn.microsoft.com/library/system.datetimeoffset(v=vs.110).aspx
TimeSpan	https://msdn.microsoft.com/library/system.timespan(v=vs.110).aspx
Byte	https://msdn.microsoft.com/library/system.byte(v=vs.110).aspx
Byte[]	https://msdn.microsoft.com/library/system.byte.aspx
Guid	GUIDs are converted to strings on input

Table of custom data types

In addition, Machine Learning Studio (classic) supports the following custom data classes.

DATA TYPE	DESCRIPTION
Data Table	The DataTable interface defines the structure of all datasets used in Azure Machine Learning.
ICluster interface	The ICluster interface defines the structure of clustering models.
IFilter interface	The IFilter interface defines the structure of digital signal processing filters applied to an entire series of numerical values. Filters can be created and then saved and applied to a new series.
ILearner interface	The ILearner interface provides a generic structure for defining and saving analytical models, excluding some special types such as clustering models.
ITransform interface	The ITransform interface provides a generic structure for defining and saving transformations. You can create an ITransform using Machine Learning Studio (classic) and then apply the transformation to new datasets.

See also

[Machine Learning Studio \(classic\)](#)

Data Table

3/10/2021 • 2 minutes to read • [Edit Online](#)

Data Table Class

A dataset is data that has been uploaded to Azure Machine Learning Studio (classic) so that it can be used in the modeling process. Even if you upload data in another format, or specify a storage format such as CSV, ARFF, or TSV, the data is implicitly converted to a `DataTable` object whenever used by a module in an experiment.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The dataset is based on the .NET [Data Table](#)

Column types

A `DataTable` consists of a collection of columns with associated metadata. These columns implement the `IArray` interface. Columns of data in Machine Learning Studio (classic) are understood to be one-dimensional arrays – that is, *vectors*.

The .NET `Array` class implements these generic interfaces: `System.Collections.Generic.IList<T>`, `System.Collections.Generic.ICollection<T>`, and `System.Collections.Generic.IEnumerable<T>`.

Columns of types `int`, `double`, and `Boolean` are typically represented as numeric dense arrays. If a dense column contains missing values, it will handled either as a missing values array or as a nullable object dense array.

Columns containing strings are handled as object dense arrays. If there are missing values, the missing values are represented either as nulls or as the type `MissingValuesObjectArray<string>`.

For more information, see [Array Class \(MSDN Library\)](#).

Getting columns in a DataTable

You can get a column by calling the `GetColumn` method on the `DataTable`. The `GetColumn` method has two overloads:

- `GetColumn(<Int64>)` gets a column by its index.
- `GetColumn(<string>)` gets a column by its name.

Other interfaces in Studio (classic)

This section also describes the following interfaces for Azure Machine Learning:

TYPE	DESCRIPTION
ICluster interface	The ICluster interface defines the structure of clustering models.
IFilter interface	The IFilter interface defines the structure of digital signal processing filters applied to an entire series of numerical values. Filters can be created and then saved and applied to a new series.
ILearner interface	The ILearner interface provides a generic structure for defining and saving analytical models, excluding some special types such as clustering models.

TYPE	DESCRIPTION
ITransform interface	The ITransform interface provides a generic structure for defining and saving transformations. You can create an iTransform using Machine Learning Studio (classic) and then apply the transformation to new datasets.

See also

[Module Data Types](#)

ICluster interface

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes `Icluster`, which is the interface for trained clustering models that is used in Azure Machine Learning Studio (classic).

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The `Icluster` interface provides methods and properties that are used to configure and interact with clustering models. A learner is defined as a set of instructions that perform standardized machine learning tasks.

The `Icluster` interface provides the following methods and properties for working with clustering models:

- Gets or sets the feature attributes
- Trains a clustering model from data
- Applies a clustering model to new data

You can interact with `Icluster` only in Studio (classic), or in one of the supported APIs.

For classification or regression models, use the `iLearner` interface.

See also

[Module data types](#)

IFilter interface

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes `IFilter`, which is the interface for working with digital signal filters in Azure Machine Learning Studio (classic).

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The `IFilter` interface provides methods and properties that are used to configure and interact with digital signal filters that have been defined using one of the filter modules in Studio (classic). For more information, see [Filter](#).

You use the `IFilter` interface to save a filter or apply a predefined filter to data.

- Specify a filter to use: its type, coefficients, etc.
- Apply the filter to input data
- Generate a `DataTable` of data with filter results

You can interact with `IFilter` only in Studio (classic), or in one of the supported APIs.

See also

[Module data types](#)

ILearner interface

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes `iLearner`, which is the interface for trained models that is used in Azure Machine Learning Studio (classic).

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The `ILearner` interface provides methods and properties that are used to configure and interact with machine learning models. A learner is defined as a set of instructions that perform standardized machine learning tasks. Learners include classification algorithms, clustering algorithms, and regression algorithms.

You can interact with `iLearner` only in Studio (classic), or in one of the supported APIs.

Studio (classic) uses this interface for the following functionality:

- Determines whether a model has the correct format.
- Gets the capabilities of the learner. These are any general properties of the learner that are not captured by the type signature of the specific learner.
- Gets or sets the settings of the learner. The settings are unique to each learner and must be configured once before any query methods can be called on the learner.

For a list of learners provided by Azure Machine Learning Studio (classic), see [Initialize Model](#).

NOTE

The [ICluster interface](#) is also available, for clustering models only.

See also

[Module data types](#)

ITransform interface

3/10/2021 • 2 minutes to read • [Edit Online](#)

This article describes `ITransform`, which is an interface in Azure Machine Learning Studio (classic) that stores a predefined transformation, or applies a predefined transformation to data.

NOTE

Applies to: Machine Learning Studio (classic)

This content pertains only to Studio (classic). Similar drag and drop modules have been added to Azure Machine Learning designer. Learn more in [this article comparing the two versions](#).

The `ITransform` interface provides the following functionality:

- Lets you save a transformation resulting from the operation of another module
- Accepts a predefined transformation
- Accepts an input dataset
- Returns a `DataTable` containing the transformed data

You can interact with `ITransform` only in Studio (classic), or in one of the supported APIs.

See also

[Module data types](#)

Troubleshoot module exceptions in Azure Machine Learning using error codes

3/10/2021 • 85 minutes to read • [Edit Online](#)

Learn about the error messages and exception codes you might encounter using modules in Azure Machine Learning Studio (classic).

To resolve the issue, look for the error in this article to read about common causes. There are two ways to get the full text of an error message in Studio (classic):

- Click the link, **View Output Log**, in the right pane and scroll to the bottom. The detailed error message is displayed in the last two lines of the window.
- Select the module that has the error, and click the red X. Only the pertinent error text is displayed.

If the error message text is not helpful, send us information about the context and any desired additions or changes. You can either submit feedback on the error topic, or visit the [Azure Machine Learning STUDIO forum](#) and post a question.

Error 0001

Exception occurs if one or more specified columns of data set couldn't be found.

You will receive this error if a column selection is made for a module, but the selected column(s) do not exist in the input data set. This error may occur if you have manually typed in a column name or if the column selector has provided a suggested column that did not exist in your dataset when you ran the experiment.

Resolution: Revisit the module throwing this exception and validate that the column name or names are correct and that all referenced columns do exist.

EXCEPTION MESSAGES

One or more specified columns were not found

Column with name or index "{0}" not found

Column with name or index "{0}" does not exist in "{1}"

Error 0002

Exception occurs if one or more parameters could not be parsed or converted from specified type into required by target method type.

This error occurs in Azure Machine Learning when you specify a parameter as input and the value type is different from the type that is expected, and implicit conversion cannot be performed.

Resolution: Check the module requirements and determine which value type is required (string, integer, double, etc.)

EXCEPTION MESSAGES

Failed to parse parameter

Failed to parse "{0}" parameter

EXCEPTION MESSAGES

Failed to parse (convert) "{0}" parameter to "{1}"

Failed to convert "{0}" parameter from "{1}" to "{2}"

Failed to convert "{0}" parameter value "{1}" from "{2}" to "{3}"

Failed to convert value "{0}" in column "{1}" from "{2}" to "{3}" with usage of the format "{4}" provided

Error 0003

Exception occurs if one or more of inputs are null or empty.

You will receive this error in Azure Machine Learning if any inputs or parameters to a module are null or empty. This error might occur, for example, when you did not type in any value for a parameter. It can also happen if you chose a dataset that has missing values, or an empty dataset.

Resolution:

- Open the module that produced the exception and verify that all inputs have been specified. Ensure that all required inputs are specified.
- Make sure that data that is loaded from Azure storage is accessible, and that the account name or key has not changed.
- Check the input data for missing values, or nulls.
- If using a query on a data source, verify that data is being returned in the format you expect.
- Check for typos or other changes in the specification of data.

EXCEPTION MESSAGES

One or more of inputs are null or empty

Input "{0}" is null or empty

Error 0004

Exception occurs if parameter is less than or equal to specific value.

You will receive this error in Azure Machine Learning if the parameter in the message is below a boundary value required for the module to process the data.

Resolution: Revisit the module throwing the exception and modify the parameter to be greater than the specified value.

EXCEPTION MESSAGES

Parameter should be greater than boundary value.

Parameter "{0}" value should be greater than {1}.

Parameter "{0}" has value "{1}" which should be greater than {2}

Error 0005

Exception occurs if parameter is less than a specific value.

You will receive this error in Azure Machine Learning if the parameter in the message is below or equal to a boundary value required for the module to process the data.

Resolution: Revisit the module throwing the exception and modify the parameter to be greater than or equal to the specified value.

EXCEPTION MESSAGES

Parameter should be greater than or equal to boundary value.

Parameter "{0}" value should be greater than or equal to {1}.

Parameter "{0}" has value "{1}" which should be greater than or equal to {2}.

Error 0006

Exception occurs if parameter is greater than or equal to the specified value.

You will receive this error in Azure Machine Learning if the parameter in the message is greater than or equal to a boundary value required for the module to process the data.

Resolution: Revisit the module throwing the exception and modify the parameter to be less than the specified value.

EXCEPTION MESSAGES

Parameters mismatch. One of the parameters should be less than another.

Parameter "{0}" value should be less than parameter "{1}" value.

Parameter "{0}" has value "{1}" which should be less than {2}.

Error 0007

Exception occurs if parameter is greater than a specific value.

You will receive this error in Azure Machine Learning if, in the properties for the module, you specified a value that is greater than is allowed. For example, you might specify a data that is outside the range of supported dates, or you might indicate that five columns be used when only three columns are available.

You might also see this error if you are specifying two sets of data that need to match in some way. For example, if you are renaming columns, and specify the columns by index, the number of names you supply must match the number of column indices. Another example might be a math operation that uses two columns, where the columns must have the same number of rows.

Resolution:

- Open the module in question and review any numeric property settings.
- Ensure that any parameter values fall within the supported range of values for that property.
- If the module takes multiple inputs, ensure that inputs are of the same size.
- If the module has multiple properties that can be set, ensure that related properties have appropriate values. For example, when using [Group Data into Bins](#), if you use the option to specify custom bin edges, the number of bins must match the number of values you provide as bin boundaries.
- Check whether the dataset or data source has changed. Sometimes a value that worked with a previous version of the data will fail after the number of columns, the column data types, or the size of the data has changed.

EXCEPTION MESSAGES

Parameters mismatch. One of the parameters should be less than or equal to another.

Parameter "{0}" value should be less than or equal to parameter "{1}" value.

Parameter "{0}" has value "{1}" which should be less than or equal to {2}.

Error 0008

Exception occurs if parameter is not in range.

You will receive this error in Azure Machine Learning if the parameter in the message is outside the bounds required for the module to process the data.

For example, this error is displayed if you try to use [Add Rows](#) to combine two datasets that have a different number of columns.

Resolution: Revisit the module throwing the exception and modify the parameter to be within the specified range.

EXCEPTION MESSAGES

Parameter value is not in the specified range.

Parameter "{0}" value is not in range.

Parameter "{0}" value should be in the range of [{1}, {2}].

Error 0009

Exception occurs when the Azure storage account name or container name is specified incorrectly.

This error occurs in Azure Machine Learning Studio (classic) when you specify parameters for an Azure storage account, but the name or password cannot be resolved. Errors on passwords or account names can happen for many reasons:

- The account is the wrong type. Some new account types are not supported for use with Machine Learning Studio (classic). See [Import Data](#) for details.
- You entered the incorrect account name
- The account no longer exists
- The password for the storage account is wrong or has changed
- You didn't specify the container name, or the container does not exist
- You didn't fully specify the file path (path to the blob)

Resolution:

Such problems often occur when you try to manually enter the account name, password, or container path. We recommend that you use the new wizard for the [Import Data](#) module, which helps you look up and check names.

Also check whether the account, container, or blob has been deleted. Use another Azure storage utility to verify that the account name and password have been entered correctly, and that the container exists.

Some newer account types are not supported by Azure Machine Learning. For example, the new "hot" or "cold" storage types cannot be used for machine learning. Both classic storage accounts and storage accounts created as "General purpose" work fine.

If the complete path to a blob was specified, verify that the path is specified as **container/blobname**, and that both the container and the blob exist in the account.

The path should not contain a leading slash. For example **/container/blob** is incorrect and should be entered as **container/blob**.

Resources

See this article for an explanation of the different storage options that are supported: [Import data into Azure Machine Learning Studio \(classic\) from various online data sources with the Import Data module](#)

Sample experiments

See these experiments in the [Cortana Intelligence Gallery](#) for examples of how to connect to different data sources:

- Input data from various sources: This lab provides a visual guide to using many of the Azure ML data sources: [AzureML experiments and data interaction](#)
- Azure Cosmos DB: [Reading data from Azure Cosmos DB in Azure Machine Learning](#)
- Import otherwise unreadable data using Python: [Load non-text file from Azure Blob storage](#)

EXCEPTION MESSAGES

The Azure storage account name or container name is incorrect.

The Azure storage account name "{0}" or container name "{1}" is incorrect; a container name of the format container/blob was expected.

Error 0010

Exception occurs if input datasets have column names that should match but do not.

You will receive this error in Azure Machine Learning if the column index in the message has different column names in the two input datasets.

Resolution: Use [Edit Metadata](#) or modify the original dataset to have the same column name for the specified column index.

EXCEPTION MESSAGES

Columns with corresponding index in input datasets have different names.

Column names are not the same for column {0} (zero-based) of input datasets {{1} and {2} respectively).

Error 0011

Exception occurs if passed column set argument does not apply to any of dataset columns.

You will receive this error in Azure Machine Learning if the specified column selection does not match any of the columns in the given dataset.

You can also get this error if you haven't selected a column and at least one column is required for the module to work.

Resolution: Modify the column selection in the module so that it will apply to the columns in the dataset.

If the module requires that you select a specific column, such as a label column, verify that the right column is selected.

If inappropriate columns are selected, remove them and rerun the experiment.

EXCEPTION MESSAGES

Specified column set does not apply to any of dataset columns.

Specified column set "{0}" does not apply to any of dataset columns.

Error 0012

Exception occurs if instance of class could not be created with passed set of arguments.

Resolution: This error is not actionable by the user and will be deprecated in a future release.

EXCEPTION MESSAGES

Untrained model, train model first.

Untrained model {{0}}, use trained model.

Error 0013

Exception occurs if the learner passed to the module is an invalid type.

This error occurs whenever a trained model is incompatible with the connected scoring module. For example, connecting the output of [Train Matchbox Recommender](#) to [Score Model](#) (instead of [Score Matchbox Recommender](#)) will generate this error when the experiment is run.

Resolution:

Determine the type of learner that is produced by the training module, and determine the scoring module that is appropriate for the learner.

If the model was trained using any of the specialized training modules, connect the trained model only to the corresponding specialized scoring module.

MODEL TYPE	TRAINING MODULE	SCORING MODULE
any classifier	Train Model or Tune Model Hyperparameters	Score Model
any regression model	Train Model or Tune Model Hyperparameters	Score Model
clustering models	Train Clustering Model or Sweep Clustering	Assign Data to Clusters
anomaly detection - One-Class SVM	Train Anomaly Detection Model	Score Model
anomaly detection - PCA	Train Model or Tune Model Hyperparameters	Score Model Some additional steps are required to evaluate the model.
anomaly detection - time series	Time Series Anomaly Detection	Model trains from data and generates scores. The module does not create a trained learner and no additional scoring is required.
recommendation model	Train Matchbox Recommender	Score Matchbox Recommender

MODEL TYPE	TRAINING MODULE	SCORING MODULE
image classification	Pretrained Cascade Image Classification	Score Model
Vowpal Wabbit models	Train Vowpal Wabbit Version 7-4 Model	Score Vowpal Wabbit Version 7-4 Model
Vowpal Wabbit models	Train Vowpal Wabbit Version 7-10 Model	Score Vowpal Wabbit Version 7-10 Model
Vowpal Wabbit models	Train Vowpal Wabbit Version 8 Model	Score Vowpal Wabbit Version 8 Model

EXCEPTION MESSAGES
Learner of invalid type is passed.
Learner "{0}" has invalid type.

Error 0014

Exception occurs if the count of column unique values is greater than allowed.

This error occurs when a column contains too many unique values. For example, you might see this error if you specify that a column be handled as categorical data, but there are too many unique values in the column to allow processing to complete. You might also see this error if there is a mismatch between the number of unique values in two inputs.

Resolution:

Open the module that generated the error, and identify the columns used as inputs. For some modules, you can right-click the dataset input and select **Visualize** to get statistics on individual columns, including the number of unique values and their distribution.

For columns that you intend to use for grouping or categorization, take steps to reduce the number of unique values in columns. You can reduce in different ways, depending on the data type of the column.

- For text data, you might be able to use [Preprocess Text](#) to collapse similar entries.
- For numeric data, you can create a smaller number of bins using [Group Data into Bins](#), remove or truncate values using [Clip Values](#), or use machine learning methods such as [Principal Component Analysis](#) or [Learning with Counts](#) to reduce the dimensionality of the data.

TIP

Unable to find a resolution that matches your scenario? You can provide feedback on this topic that includes the name of the module that generated the error, and the data type and cardinality of the column. We will use the information to provide more targeted troubleshooting steps for common scenarios.

EXCEPTION MESSAGES
Number of column unique values is greater than allowed.
Number of unique values in column: "{0}" exceeds tuple count of {1}.

Error 0015

Exception occurs if database connection has failed.

You will receive this error if you enter an incorrect SQL account name, password, database server, or database name, or if a connection with the database cannot be established due to problems with the database or server.

Resolution: Verify that the account name, password, database server, and database have been entered correctly, and that the specified account has the correct level of permissions. Verify that the database is currently accessible.

EXCEPTION MESSAGES

Error making database connection.

Error making database connection: {0}.

Error 0016

Exception occurs if input datasets passed to the module should have compatible column types but do not.

You will receive this error in Azure Machine Learning if the types of the columns passed in two or more datasets are not compatible with each other.

Resolution: Use [Edit Metadata](#), modify the original input dataset, or use [Convert to Dataset](#) to ensure that the types of the columns are compatible.

EXCEPTION MESSAGES

Columns with corresponding index in input datasets do have incompatible types.

Columns {0} and {1} are incompatible.

Column element types are not compatible for column {0} (zero-based) of input datasets ({1} and {2} respectively).

Error 0017

Exception occurs if a selected column uses a data type that is not supported by the current module.

For example, you might receive this error in Azure Machine Learning if your column selection includes a column with a data type that cannot be processed by the module, such as a string column for a math operation, or a score column where a categorical feature column is required.

Resolution:

1. Identify the column that is the problem.
2. Review the requirements of the module.
3. Modify the column to make it conform to requirements. You might need to use several of the following modules to make changes, depending on the column and the conversion you are attempting:
 - Use [Edit Metadata](#) to change the data type of columns, or to change the column usage from feature to numeric, categorical to non-categorical, and so forth.
 - Use [Convert to Dataset](#) to ensure that all included columns use data types that are supported by Azure Machine Learning. If you cannot convert the columns, consider removing them from the input dataset.
 - Use the [Apply SQL Transformation](#) or [Execute R Script](#) modules to cast or convert any columns that cannot be modified using [Edit Metadata](#). These modules provide more flexibility for working with datetime data types.
 - For numeric data types, you can use the [Apply Math Operation](#) module to round or truncate values, or use the [Clip Values](#) module to remove out of range values.

4. As a last resort, you might need to modify the original input dataset.

TIP

Unable to find a resolution that matches your scenario? You can provide feedback on this topic that includes the name of the module that generated the error, and the data type and cardinality of the column. We will use the information to provide more targeted troubleshooting steps for common scenarios.

EXCEPTION MESSAGES

Cannot process column of current type. The type is not supported by the module.

Cannot process column of type {0}. The type is not supported by the module.

Cannot process column "{1}" of type {0}. The type is not supported by the module.

Cannot process column "{1}" of type {0}. The type is not supported by the module. Parameter name: {2}

Error 0018

Exception occurs if input dataset is not valid.

Resolution: This error in Azure Machine Learning can appear in many contexts, so there is not a single resolution. In general, the error indicates that the data provided as input to a module has the wrong number of columns, or that the data type does not match requirements of the module. For example:

- The module requires a label column, but no column is marked as a label, or you have not selected a label column yet.
- The module requires that data be categorical but your data is numeric.
- The module requires a specific data type. For example, ratings provided to [Train Matchbox Recommender](#) can be either numeric or categorical, but cannot be floating point numbers.
- The data is in the wrong format.
- Imported data contains invalid characters, bad values, or out of range values.
- The column is empty or contains too many missing values.

To determine the requirements and how your data might, review the help topic for the module that will be consuming the dataset as input.

We also recommend that you use [Summarize Data](#) or [Compute Elementary Statistics](#) to profile your data, and use these modules to fix metadata and clean values: [Edit Metadata](#), [Clean Missing Data](#), [Clip Values](#).

EXCEPTION MESSAGES

Dataset is not valid.

{0} contains invalid data.

{0} and {1} should be consistent column wise.

Error 0019

Exception occurs if column is expected to contain sorted values, but it does not.

You will receive this error in Azure Machine Learning if the specified column values are out of order.

Resolution: Sort the column values by manually modifying the input dataset and rerun the module.

EXCEPTION MESSAGES

Values in column are not sorted.

Values in column "{0}" are not sorted.

Values in column "{0}" of dataset "{1}" are not sorted.

Error 0020

Exception occurs if number of columns in some of the datasets passed to the module is too small.

You will receive this error in Azure Machine Learning if not enough columns have been selected for a module.

Resolution: Revisit the module and ensure that column selector has correct number of columns selected.

EXCEPTION MESSAGES

Number of columns in input dataset is less than allowed minimum.

Number of columns in input dataset is less than allowed minimum of {0} column(s).

Number of columns in input dataset "{0}" is less than allowed minimum of {1} column(s).

Error 0021

Exception occurs if number of rows in some of the datasets passed to the module is too small.

This error is seen in Azure Machine Learning when there are not enough rows in the dataset to perform the specified operation. For example, you might see this error if the input dataset is empty, or if you are trying to perform an operation that requires some minimum number of rows to be valid. Such operations can include (but are not limited to) grouping or classification based on statistical methods, certain types of binning, and learning with counts.

Resolution:

- Open the module that returned the error, and check the input dataset and module properties.
- Verify that the input dataset is not empty and that there are enough rows of data to meet the requirements described in module help.
- If your data is loaded from an external source, make sure that the data source is available and that there is no error or change in the data definition that would cause the import process to get fewer rows.
- If you are performing an operation on the data upstream of the module that might affect the type of data or the number of values, such as cleaning, splitting, or join operations, check the outputs of those operations to determine the number of rows returned.

Error 0022

Exception occurs if number of selected columns in input dataset does not equal to the expected number.

This error in Azure Machine Learning can occur when the downstream module or operation requires a specific number of columns or inputs, and you have provided too few or too many columns or inputs. For example:

- You specify a single label column or key column and accidentally selected multiple columns.
- You are renaming columns, but provided more or fewer names than there are columns.
- The number of columns in the source or destination has changed or doesn't match the number of

columns used by the module.

- You have provided a comma-separated list of values for inputs, but the number of values does not match, or multiple inputs are not supported.

Resolution: Revisit the module and check the column selection to ensure that the correct number of columns is selected. Verify the outputs of upstream modules, and the requirements of downstream operations.

If you used one of the column selection options that can select multiple columns (column indices, all features, all numeric, etc.), validate the exact number of columns returned by the selection.

If you are trying to specify a comma-separated list of datasets as inputs to [Unpack Zipped Datasets](#), unpack only one dataset at a time. Multiple inputs are not supported.

Verify that the number or type of upstream columns has not changed.

If you are using a recommendation dataset to train a model, remember that the recommender expects a limited number of columns, corresponding to user-item pairs or user-item-rankings. Remove additional columns before training the model or splitting recommendation datasets. For more information, see [Split Data](#).

EXCEPTION MESSAGES

Number of selected columns in input dataset does not equal to the expected number.

Number of selected columns in input dataset does not equal to {0}.

Column selection pattern "{0}" provides number of selected columns in input dataset not equal to {1}.

Column selection pattern "{0}" is expected to provide {1} column(s) selected in input dataset, but {2} column(s) is/are provided.

Error 0023

Exception occurs if target column of input dataset is not valid for the current trainer module.

This error in Azure Machine Learning occurs if the target column (as selected in the module parameters) is not of the valid data-type, contained all missing values, or was not categorical as expected.

Resolution: Revisit the module input to inspect the content of the label/target column. Make sure it does not have all missing values. If the module is expecting target column to be categorical, make sure that there are more than one distinct values in the target column.

EXCEPTION MESSAGES

Input dataset has unsupported target column.

Input dataset has unsupported target column "{0}".

Input dataset has unsupported target column "{0}" for learner of type {1}.

Error 0024

Exception occurs if dataset does not contain a label column.

This error in Azure Machine Learning occurs when the module requires a label column and the dataset does not have a label column. For example, evaluation of a scored dataset usually requires that a label column is present to compute accuracy metrics.

It can also happen that a label column is present in the dataset, but not detected correctly by Azure Machine Learning.

Resolution:

- Open the module that generated the error, and determine if a label column is present. The name or data type of the column doesn't matter, as long as the column contains a single outcome (or dependent variable) that you are trying to predict. If you are not sure which column has the label, look for a generic name such as *Class* or *Target*.
- If the dataset does not include a label column, it is possible that the label column was explicitly or accidentally removed upstream. It could also be that the dataset is not the output of an upstream scoring module.
- To explicitly mark the column as the label column, add the [Edit Metadata](#) module and connect the dataset. Select only the label column, and select **Label** from the **Fields** dropdown list.
- If the wrong column is chosen as the label, you can select **Clear label** from the **Fields** to fix the metadata on the column.

EXCEPTION MESSAGES

There is no label column in dataset.

There is no label column in "{0}".

Error 0025

Exception occurs if dataset does not contain a score column.

This error in Azure Machine Learning occurs if the input to the evaluate model does not contain valid score columns. For example, the user attempts to evaluate a dataset before it was scored with a correct trained model, or the score column was explicitly dropped upstream. This exception also occurs if the score columns on the two datasets are incompatible. For example, you might be trying to compare the accuracy of a linear regressor with that of a binary classifier.

Resolution: Revisit the input to the evaluate model and examine if it contains one or more score columns. If not, the dataset was not scored or the score columns were dropped in an upstream module.

EXCEPTION MESSAGES

There is no score column in dataset.

There is no score column in "{0}".

There is no score column in "{0}" that is produced by a "{1}". Score the dataset using the correct type of learner.

Error 0026

Exception occurs if columns with the same name are not allowed.

This error in Azure Machine Learning occurs if multiple columns have the same name. One way you may receive this error is if the dataset does not have a header row and column names are automatically assigned: Col0, Col1, etc.

Resolution: If columns have same name, insert a [Edit Metadata](#) module between the input dataset and the module. Use the column selector in [Edit Metadata](#) to select columns to rename, typing the new names into the **New column names** textbox.

EXCEPTION MESSAGES

Equal column names are specified in arguments. Equal column names are not allowed by module.

EXCEPTION MESSAGES

Equal column names in arguments "{0}" and "{1}" are not allowed. Specify different names.

Error 0027

Exception occurs in case when two objects have to be of the same size but are not.

This is a common error in Azure Machine Learning and can be caused by many conditions.

Resolution: There is no specific resolution. However, you can check for conditions such as the following:

- If you are renaming columns, make sure that each list (the input columns and the list of new names) has the same number of items.
- If you are joining or concatenating two datasets, make sure they have the same schema.
- If you are joining two datasets that have multiple columns, make sure that the key columns have the same data type, and select the option **Allow duplicates and preserve column order in selection**.

EXCEPTION MESSAGES

The size of passed objects is inconsistent.

The size of "{0}" is inconsistent with size of "{1}".

Error 0028

Exception occurs in the case when column set contains duplicated column names and it is not allowed.

This error in Azure Machine Learning occurs when column names are duplicated; that is, not unique.

Resolution: If any columns have same name, add an instance of [Edit Metadata](#) between the input dataset and the module raising the error. Use the Column Selector in [Edit Metadata](#) to select columns to rename, and type the new column names into the **New column names** textbox. If you are renaming multiple columns, ensure that the values you type in the **New column names** are unique.

EXCEPTION MESSAGES

Column set contains duplicated column name(s).

The name "{0}" is duplicated.

The name "{0}" is duplicated in "{1}".

Error 0029

Exception occurs in case when invalid URI is passed.

This error in Azure Machine Learning occurs in case when invalid URI is passed. You will receive this error if any of the following conditions are true; or.

- The Public or SAS URI provided for Azure Blob Storage for read or write contains an error.
- The time window for the SAS has expired.
- The Web URL via HTTP source represents a file or a loopback URI.
- The Web URL via HTTP contains an incorrectly formatted URL.

- The URL cannot be resolved by the remote source.

Resolution: Revisit the module and verify the format of the URI. If the data source is a Web URL via HTTP, verify that the intended source is not a file or a loopback URI (localhost).

EXCEPTION MESSAGES

Invalid Uri is passed.

Error 0030

Exception occurs in the case when it is not possible to download a file.

This exception in Azure Machine Learning occurs when it is not possible to download a file. You will receive this exception when an attempted read from an HTTP source has failed after three (3) retry attempts.

Resolution: Verify that the URI to the HTTP source is correct and that the site is currently accessible via the Internet.

EXCEPTION MESSAGES

Unable to download a file.

Error while downloading the file: {0}.

Error 0031

Exception occurs if number of columns in column set is less than needed.

This error in Azure Machine Learning occurs if the number of columns selected is less than needed. You will receive this error if the minimum required number of columns are not selected.

Resolution: Add additional columns to the column selection by using the **Column Selector**.

EXCEPTION MESSAGES

Number of columns in column set is less than required.

{0} column(s) should be specified. The actual number of specified columns is {1}.

Error 0032

Exception occurs if argument is not a number.

You will receive this error in Azure Machine Learning if the argument is a double or NaN.

Resolution: Modify the specified argument to use a valid value.

EXCEPTION MESSAGES

Argument is not a number.

"{0}" is not a number.

Error 0033

Exception occurs if argument is Infinity.

This error in Azure Machine Learning occurs if the argument is infinite. You will receive this error if the argument is either `double.NegativeInfinity` or `double.PositiveInfinity`.

Resolution: Modify the specified argument to be a valid value.

EXCEPTION MESSAGES

Argument is must be finite.

"{0}" is not finite.

Error 0034

Exception occurs if more than one rating exists for a given user-item pair.

This error in Azure Machine Learning occurs in recommendation if a user-item pair has more than one rating value.

Resolution: Ensure that the user-item pair possesses one rating value only.

EXCEPTION MESSAGES

More than one rating exists for the value(s) in dataset.

More than one rating for user {0} and item {1} in rating prediction data table.

Error 0035

Exception occurs if no features were provided for a given user or item.

This error in Azure Machine Learning occurs you are trying to use a recommendation model for scoring but a feature vector cannot be found.

Resolution:

The Matchbox recommender has certain requirements that must be met when using either item features or user features. This error indicates that a feature vector is missing for a user or item that you provided as input. You must ensure that a vector of features is available in the data for each user or item.

For example, if you trained a recommendation model using features such as the user's age, location, or income, but now want to create scores for new users who were not seen during training, you must provide some equivalent set of features (namely, age, location, and income values) for the new users in order to make appropriate predictions for them.

If you do not have any features for these users, consider feature engineering to generate appropriate features. For example, if you do not have individual user age or income values, you might generate approximate values to use for a group of users.

When you are scoring from a recommendation mode, you can use item or user features only if you previously used item or user features during training. For more information, see [Score Matchbox Recommender](#).

For general information about how the Matchbox recommendation algorithm works, and how to prepare a dataset of item features or user features, see [Train Matchbox Recommender](#).

TIP

Resolution not applicable to your case? You are welcome to send feedback on this article and provide information about the scenario, including the module and the number of rows in the column. We will use this information to provide more detailed troubleshooting steps in the future.

EXCEPTION MESSAGES

No features were provided for a required user or item.

Features for {0} required but not provided.

Error 0036

Exception occurs if multiple feature vectors were provided for a given user or item.

This error in Azure Machine Learning occurs if a feature vector is defined more than once.

Resolution: Ensure that the feature vector is not defined more than once.

EXCEPTION MESSAGES

Duplicate feature definition for a user or item.

Duplicate feature definition for {0}.

Error 0037

Exception occurs if multiple label columns are specified and just one is allowed.

This error in Azure Machine Learning occurs if more than one column is selected to be the new label column. Most supervised learning algorithms require a single column to be marked as the target or label.

Resolution: Make sure to select a single column as the new label column.

EXCEPTION MESSAGES

Multiple label columns are specified.

Error 0038

Exception occurs if number of elements expected should be an exact value, but is not.

This error in Azure Machine Learning occurs if the number of elements expected should be an exact value, but is not. You will receive this error if the number of elements is not equal to the valid expected value.

Resolution: Modify the input to have the correct number of elements.

EXCEPTION MESSAGES

Number of elements is not valid.

Number of elements in "{0}" is not valid.

Number of elements in "{0}" is not equal to valid number of {1} element(s).

Error 0039

Exception occurs if an operation has failed.

This error in Azure Machine Learning occurs when an internal operation cannot be completed.

Resolution: This error is caused by many conditions and there is no specific remedy.

The following table contains generic messages for this error, which are followed by a specific description of the

condition.

If no details are available, [send feedback](#) and provide information about the modules that generated the error and related conditions.

EXCEPTION MESSAGES

Operation failed.

Error while completing operation: {0}.

Error 0040

Exception occurs when calling a deprecated module.

This error in Azure Machine Learning is produced when calling a deprecated module.

Resolution: Replace the deprecated module with a supported one. See the module output log for the information about which module to use instead.

EXCEPTION MESSAGES

Accessing deprecated module.

Module "{0}" is deprecated. Use module "{1}" instead.

Error 0041

Exception occurs when calling a deprecated module.

This error in Azure Machine Learning is produced when calling a deprecated module.

Resolution: Replace the deprecated module with a set of supported ones. This information should be visible in the module output log.

EXCEPTION MESSAGES

Accessing deprecated module.

Module "{0}" is deprecated. Use the modules "{1}" for requested functionality.

Error 0042

Exception occurs when it is not possible to convert column to another type.

This error in Azure Machine Learning occurs when it is not possible to convert column to the specified type. You will receive this error if a module requires a particular data type, such as datetime, text, a floating point number, or integer, but it is not possible to convert an existing column to the required type.

For example, you might select a column and try to convert it to a numeric data type for use in a math operation, and get this error if the column contained invalid data.

Another reason you might get this error if you try to use a column containing floating point numbers or many unique values as a categorical column.

Resolution:

- Open the help page for the module that generated the error, and verify the data type requirements.
- Review the data types of the columns in the input dataset.

- Inspect data originating in so-called schema-less data sources.
- Check the dataset for missing values or special characters that might block conversion to the desired data type.
 - Numeric data types should be consistent: for example, check for floating point numbers in a column of integers.
 - Look for text strings or NA values in a number column.
 - Boolean values can be converted to an appropriate representation depending on the required data type.
 - Examine text columns for non-unicode characters, tab characters, or control characters
 - Datetime data should be consistent to avoid modeling errors, but cleanup can be complex owing to the many formats. Consider using the [Execute R Script](#) or [Execute Python Script](#) modules to perform cleanup.
- If necessary, modify the values in the input dataset so that the column can be converted successfully. Modification might include binning, truncation or rounding operations, elimination of outliers, or imputation of missing values. See the following articles for some common data transformation scenarios in machine learning:
 - [Clip Values](#)
 - [Clean Missing Data](#)
 - [Normalize Data](#)
 - [Group Data Into Bins](#)

TIP

Resolution unclear, or not applicable to your case? You are welcome to send feedback on this article and provide information about the scenario, including the module and the data type of the column. We will use this information to provide more detailed troubleshooting steps in the future.

EXCEPTION MESSAGES

Not allowed conversion.

Could not convert column of type {0} to column of type {1}.

Could not convert column "{2}" of type {0} to column of type {1}.

Could not convert column "{2}" of type {0} to column "{3}" of type {1}.

Error 0043

Exception occurs when element type does not explicitly implement Equals.

This error in Azure Machine Learning is unused and will be deprecated.

Resolution: None.

EXCEPTION MESSAGES

No accessible explicit method Equals found.

Cannot compare values for column \"{0}\" of type {1}. No accessible explicit method Equals found.

Error 0044

Exception occurs when it is not possible to derive element type of column from the existing values.

This error in Azure Machine Learning occurs when it is not possible to infer the type of a column or columns in a dataset. This typically happens when concatenating two or more datasets with different element types. If Azure Machine Learning is unable to determine a common type that is able to represent all the values in a column or columns without loss of information, it will generate this error.

Resolution: Ensure that all values in a given column in both datasets being combined are either of the same type (numeric, Boolean, categorical, string, date, etc.) or can be coerced to the same type.

EXCEPTION MESSAGES

Cannot derive element type of the column.

Cannot derive element type for column "{0}" -- all the elements are null references.

Cannot derive element type for column "{0}" of dataset "{1}" -- all the elements are null references.

Error 0045

Exception occurs when it is not possible to create a column because of mixed element types in the source.

This error in Azure Machine Learning is produced when the element types of two datasets being combined are different.

Resolution: Ensure that all values in a given column in both datasets being combined are of the same type (numeric, Boolean, categorical, string, date, etc.).

EXCEPTION MESSAGES

Cannot create column with mixed element types.

Cannot create column with ID "{0}" of mixed element types:\n\tType of data[{1}, {0}] is {2}\n\tType of data[{3}, {0}] is {4}.

Error 0046

Exception occurs when it is not possible to create directory on specified path.

This error in Azure Machine Learning occurs when it is not possible to create a directory on the specified path. You will receive this error if any part of the path to the output directory for a Hive Query is incorrect or inaccessible.

Resolution: Revisit the module and verify that the directory path is correctly formatted and that it is accessible with the current credentials.

EXCEPTION MESSAGES

Specify a valid output directory.

Directory: {0} cannot be created. Specify valid path.

Error 0047

Exception occurs if number of feature columns in some of the datasets passed to the module is too small.

This error in Azure Machine Learning occurs if the input dataset to training does not contain the minimum number of columns required by the algorithm. Typically either the dataset is empty or only contains training columns.

Resolution: Revisit the input dataset to make sure there are one or more additional columns apart from the label

column.

EXCEPTION MESSAGES

Number of feature columns in input dataset is less than allowed minimum.

Number of feature columns in input dataset is less than allowed minimum of {0} column(s).

Number of feature columns in input dataset "{0}" is less than allowed minimum of {1} column(s).

Error 0048

Exception occurs in the case when it is not possible to open a file.

This error in Azure Machine Learning occurs when it is not possible to open a file for read or write. You might receive this error for these reasons:

- The container or the file (blob) does not exist
- The access level of the file or container does not allow you to access the file
- The file is too large to read or the wrong format

Resolution: Revisit the module and the file you are trying to read.

Verify that the names of the container and file are correct.

Use the Azure classic portal or an Azure storage tool to verify that you have permission to access the file.

If you are trying to read an image file, make sure that it meets the requirements for image files in terms of size, number of pixels, and so forth. For more information, see [Import Images](#).

EXCEPTION MESSAGES

Unable to open a file.

Error while opening the file: {0}.

Error 0049

Exception occurs in the case when it is not possible to parse a file.

This error in Azure Machine Learning occurs when it is not possible to parse a file. You will receive this error if the file format selected in the [Import Data](#) module does not match the actual format of the file, or if the file contains an unrecognizable character.

Resolution: Revisit the module and correct the file format selection if it does not match the format of the file. If possible, inspect the file to confirm that it does not contain any illegal characters.

EXCEPTION MESSAGES

Unable to parse a file.

Error while parsing the file: {0}.

Error 0050

Exception occurs in the case when input and output files are the same.

Resolution: This error in Azure Machine Learning is unused and will be deprecated.

EXCEPTION MESSAGES

Specified files for input and output cannot be the same.

Error 0051

Exception occurs in the case when several output files are the same.

Resolution: This error in Azure Machine Learning is unused and will be deprecated.

EXCEPTION MESSAGES

Specified files for outputs cannot be the same.

Error 0052

Exception occurs if Azure storage account key is specified incorrectly.

This error in Azure Machine Learning occurs if the key used to access the Azure storage account is incorrect. For example, you might see this error if the Azure storage key was truncated when copied and pasted, or if the wrong key was used.

For more information about how to get the key for an Azure storage account, see [View, copy, and regenerate storage access keys](#).

Resolution: Revisit the module and verify that the Azure storage key is correct for the account; copy the key again from the Azure classic portal if necessary.

EXCEPTION MESSAGES

The Azure storage account key is incorrect.

Error 0053

Exception occurs in the case when there are no user features or items for matchbox recommendations.

This error in Azure Machine Learning is produced when a feature vector cannot be found.

Resolution: Ensure that a feature vector is present in the input dataset.

EXCEPTION MESSAGES

User features or/and items are required but not provided.

Error 0054

Exception occurs if there is too few distinct values in the column to complete operation.

Resolution: This error in Azure Machine Learning is unused and will be deprecated.

EXCEPTION MESSAGES

Data has too few distinct values in the specified column to complete operation.

Data has too few distinct values in the specified column to complete operation. The required minimum is {0} elements.

EXCEPTION MESSAGES

Data has too few distinct values in the column "{1}" to complete operation. The required minimum is {0} elements.

Error 0055

Exception occurs when calling a deprecated module.

This error in Azure Machine Learning appears if you try to call a module that has been deprecated.

Resolution:

EXCEPTION MESSAGES

Accessing deprecated module.

Module "{0}" is deprecated.

Error 0056

Exception occurs if the columns you selected for an operation violates requirements.

This error in Azure Machine Learning occurs when you are choosing columns for an operation that requires the column be of a particular data type.

This error can also happen if the column is the correct data type, but the module you are using requires that the column also be marked as a feature, label, or categorical column.

For example, the [Convert to Indicator Values](#) module requires that columns be categorical, and will raise this error if you select a feature column or label column.

Resolution:

1. Review the data type of the columns that are currently selected.
2. Ascertain whether the selected columns are categorical, label, or feature columns.
3. Review the help topic for the module in which you made the column selection, to determine if there are specific requirements for data type or column usage.
4. Use [Edit Metadata](#) to change the column type for the duration of this operation. Be sure to change the column type back to its original value, using another instance of [Edit Metadata](#), if you need it for downstream operations.

EXCEPTION MESSAGES

One or more selected columns were not in an allowed category.

Column with name "{0}" is not in an allowed category.

Error 0057

Exception occurs when attempting to create a file or blob that already exists.

This exception occurs when you are using the [Export Data](#) module or other module to save results of an experiment in Azure Machine Learning to Azure blob storage, but you attempt to create a file or blob that already exists.

Resolution:

You will receive this error only if you previously set the property **Azure blob storage write mode** to **Error**. By design, this module raises an error if you attempt to write a dataset to a blob that already exists.

- Open the module properties and change the property **Azure blob storage write mode** to **Overwrite**.
- Alternatively, you can type the name of a different destination blob or file and be sure to specify a blob that does not already exist.

EXCEPTION MESSAGES

File or Blob already exists.

File or Blob "{0}" already exists.

Error 0058

This error in Azure Machine Learning occurs if the dataset does not contain the expected label column.

This exception can also occur when the label column provided does not match the data or datatype expected by the learner, or has the wrong values. For example, this exception is produced when using a real-valued label column when training a binary classifier.

Resolution: The resolution depends on the learner or trainer that you are using, and the data types of the columns in your dataset. First, verify the requirements of the machine learning algorithm or training module.

Revisit the input dataset. Verify that the column you expect to be treated as the label has the right data type for the model you are creating.

Check inputs for missing values and eliminate or replace them if necessary.

If necessary, add the [Edit Metadata](#) module and ensure that the label column is marked as a label.

EXCEPTION MESSAGES

The label column is not as expected

The label column is not as expected in "{0}".

The label column "{0}" is not expected in "{1}".

Error 0059

Exception occurs if a column index specified in a column picker cannot be parsed.

This error in Azure Machine Learning occurs if a column index specified when using the Column Selector cannot be parsed. You will receive this error when the column index is in an invalid format that cannot be parsed.

Resolution: Modify the column index to use a valid index value.

EXCEPTION MESSAGES

One or more specified column indexes or index ranges could not be parsed.

Column index or range "{0}" could not be parsed.

Error 0060

Exception occurs when an out of range column range is specified in a column picker.

This error in Azure Machine Learning occurs when an out-of-range column range is specified in the Column Selector. You will receive this error if the column range in the column picker does not correspond to the columns in the dataset.

Resolution: Modify the column range in the column picker to correspond to the columns in the dataset.

EXCEPTION MESSAGES

Invalid or out of range column index range specified.

Column range "{0}" is invalid or out of range.

Error 0061

Exception occurs when attempting to add a row to a DataTable that has a different number of columns than the table.

This error in Azure Machine Learning occurs when you attempt to add a row to a dataset that has a different number of columns than the dataset. You will receive this error if the row that is being added to the dataset has a different number of columns from the input dataset. The row cannot be appended to the dataset if the number of columns is different.

Resolution: Modify the input dataset to have the same number of columns as the row added, or modify the row added to have the same number of columns as the dataset.

EXCEPTION MESSAGES

All tables must have the same number of columns.

Error 0062

Exception occurs when attempting to compare two models with different learner types.

This error in Azure Machine Learning is produced when evaluation metrics for two different scored datasets cannot be compared. In this case, it is not possible to compare the effectiveness of the models used to produce the two scored datasets.

Resolution: Verify that the scored results are produced by the same kind of machine learning model (binary classification, regression, multi-class classification, recommendation, clustering, anomaly detection, etc.) All models that you compare must have the same learner type.

EXCEPTION MESSAGES

All models must have the same learner type.

Error 0063

This exception is raised when R script evaluation fails with an error.

This error occurs when you have provided an R script in one of the [R language modules](#) in Azure Machine Learning, and the R code contains internal syntax errors. The exception can also occur if you provide the wrong inputs to the R script.

The error can also occur if the script is too large to execute in the workspace. The maximum script size for the **Execute R Script** module is 1,000 lines or 32 KB of work space, whichever is lesser.

Resolution:

1. In Azure Machine Learning Studio (classic), right-click the module that has the error, and select **View Log**.

2. Examine the standard error log of the module, which contains the stack trace.
 - Lines beginning with [ModuleOutput] indicate output from R.
 - Messages from R marked as **warnings** typically do not cause the experiment to fail.
3. Resolve script issues.
 - Check for R syntax errors. Check for variables that are defined but never populated.
 - Review the input data and the script to determine if either the data or variables in the script use characters not supported by Azure Machine Learning.
 - Check whether all package dependencies are installed.
 - Check whether your code loads required libraries that are not loaded by default.
 - Check whether the required packages are the correct version.
 - Make sure that any dataset that you want to output is converted to a data frame.
4. Resubmit the experiment.

NOTE

These topics contain examples of R code that you can use, as well as links to experiments in the [Cortana Intelligence Gallery](#) that use R script.

- [Execute R Script](#)
- [Create R Model](#)

EXCEPTION MESSAGES

Error during evaluation of R script.

The following error occurred during evaluation of R script: ----- Start of error message from R ----- {0} -----
End of error message from R -----

During the evaluation of R script "{1}" the following error occurred: ----- Start of error message from R ----- {0} -----
----- End of error message from R -----

Error 0064

Exception occurs if Azure storage account name or storage key is specified incorrectly.

This error in Azure Machine Learning occurs if the Azure storage account name or storage key is specified incorrectly. You will receive this error if you enter an incorrect account name or password for the storage account. This may occur if you manually enter the account name or password. It may also occur if the account has been deleted.

Resolution: Verify that the account name and password have been entered correctly, and that the account exists.

EXCEPTION MESSAGES

The Azure storage account name or storage key is incorrect.

The Azure storage account name "{0}" or storage key for the account name is incorrect.

Error 0065

Exception occurs if Azure blob name is specified incorrectly.

This error in Azure Machine Learning occurs if the Azure blob name is specified incorrectly. You will receive the error if:

- The blob cannot be found in the specified container.
- The fully qualified name of the blob specified for output in one of the [Learning with Counts](#) modules is greater than 512 characters.
- Only the container was specified as the source in a [Import Data](#) request when the format was Excel or CSV with encoding; concatenation of the contents of all blobs within a container is not allowed with these formats.
- A SAS URI does not contain the name of a valid blob.

Resolution: Revisit the module throwing the exception. Verify that the specified blob does exist in the container in the storage account and that permissions allow you to see the blob. Verify that the input is of the form **containername/filename** if you have Excel or CSV with encoding formats. Verify that a SAS URI contains the name of a valid blob.

EXCEPTION MESSAGES

The Azure storage blob is incorrect.

The Azure storage blob name "{0}" is incorrect

Error 0066

Exception occurs if a resource could not be uploaded to an Azure Blob.

This error in Azure Machine Learning occurs if a resource could not be uploaded to an Azure Blob. You will receive this message if [Train Vowpal Wabbit 7-4 Model](#) encounters an error attempting to save either the model or the hash created when training the model. Both are saved to the same Azure storage account as the account containing the input file.

Resolution: Revisit the module. Verify that the Azure account name, storage key, and container are correct and that the account has permission to write to the container.

EXCEPTION MESSAGES

The resource could not be uploaded to Azure storage.

The file "{0}" could not be uploaded to Azure storage as {1}.

Error 0067

Exception occurs if a dataset has a different number of columns than expected.

This error in Azure Machine Learning occurs if a dataset has a different number of columns than expected. You will receive this error when the number of columns in the dataset are different from the number of columns that the module expects during execution.

Resolution: Modify the input dataset or the parameters.

EXCEPTION MESSAGES

Unexpected number of columns in the datatable.

Expected "{0}" columns but found "{1}" columns instead.

Error 0068

Exception occurs if the specified Hive script is not correct.

This error in Azure Machine Learning occurs if there are syntax errors in a Hive QL script, or if the Hive interpreter encounters an error while executing the query or script.

Resolution:

The error message from Hive is normally reported back in the Error Log so that you can take action based on the specific error.

- Open the module and inspect the query for mistakes.
- Verify that the query works correctly outside of Azure Machine Learning by logging in to the Hive console of your Hadoop cluster and running the query.
- Try placing comments in your Hive script in a separate line as opposed to mixing executable statements and comments in a single line.

Resources

See the following articles for help with Hive queries for machine learning:

- [Create Hive tables and load data from Azure Blob Storage](#)
- [Explore data in tables with Hive queries](#)
- [Create features for data in an Hadoop cluster using Hive queries](#)
- [Hive for SQL Users Cheat Sheet \(PDF\)](#)

EXCEPTION MESSAGES

Hive script is incorrect.

Hive script {0} is not correct.

Error 0069

Exception occurs if the specified SQL script is not correct.

This error in Azure Machine Learning occurs if the specified SQL script has syntax problems, or if the columns or table specified in the script is not valid.

You will receive this error if the SQL engine encounters any error while executing the query or script. The SQL error message is normally reported back in the Error Log so that you can take action based on the specific error.

Resolution:

Revisit the module and inspect the SQL query for mistakes.

Verify that the query works correctly outside of Azure ML by logging in to the database server directly and running the query.

If there is a SQL generated message reported by the module exception, take action based on the reported error. For example, the error messages sometimes include specific guidance on the likely error:

- *No such column or missing database*, indicating that you might have typed a column name wrong. If you are sure the column name is correct, try using brackets or quotation marks to enclose the column identifier.
- *SQL logic error near <SQL keyword>*, indicating that you might have a syntax error before the specified keyword

EXCEPTION MESSAGES

SQL script is incorrect.

SQL query "{0}" is not correct.

EXCEPTION MESSAGES

SQL query "{0}" is not correct: {1}

Error 0070

Exception occurs when attempting to access non-existent Azure table.

This error in Azure Machine Learning occurs when you attempt to access a non-existent Azure table. You will receive this error if you specify a table in Azure storage, which does not exist when reading from or writing to Azure Table Storage. This can happen if you mistype the name of the desired table, or you have a mismatch between the target name and the storage type. For example, you intended to read from a table but entered the name of a blob instead.

Resolution: Revisit the module to verify that the name of the table is correct.

EXCEPTION MESSAGES

Azure table does not exist.

Azure table "{0}" does not exist.

Error 0071

Exception occurs if provided credentials are incorrect.

This error in Azure Machine Learning occurs if the provided credentials are incorrect.

You might also receive this error if the module cannot connect to an HDInsight cluster.

Resolution: Review the inputs to the module and verify the account name and password.

Check for the following issues that can cause an error:

- The schema of the dataset does not match the schema of the destination datatable.
- Column names are missing or misspelled
- You are writing to a table that has column names with illegal characters. Ordinarily you can enclose such column names in square brackets, but if that does not work, edit column names to use only letters and underscores (_)
- Strings that you are trying to write contain single quotation marks

If you are trying to connect to an HDInsight cluster, verify that the target cluster is accessible with the supplied credentials.

EXCEPTION MESSAGES

Incorrect credentials are passed.

Incorrect username "{0}" or password is passed

Error 0072

Exception occurs in the case of connection timeout.

This error in Azure Machine Learning occurs when a connection times out. You will receive this error if there are currently connectivity issues with the data source or destination, such as slow internet connectivity, or if the

dataset is large and/or the SQL query to read in the data performs complicated processing.

Resolution: Determine whether there are currently issues with slow connections to Azure storage or the internet.

EXCEPTION MESSAGES

Connection timeout occurred.

Error 0073

Exception occurs if an error occurs while converting a column to another type.

This error in Azure Machine Learning occurs when it is not possible to convert column to another type. You will receive this error if a module requires a particular type and it is not possible to convert the column to the new type.

Resolution: Modify the input dataset so that the column can be converted based on the inner exception.

EXCEPTION MESSAGES

Failed to convert column.

Failed to convert column to {0}.

Error 0074

Exception occurs when the [Edit Metadata](#) tries to convert a sparse column to categorical.

This error in Azure Machine Learning occurs when the [Edit Metadata](#) tries to convert a sparse column to categorical. You will receive this error when trying to convert sparse columns to categorical with the **Make categorical** option. Azure machine Learning does not support sparse categorical arrays, so the module will fail.

Resolution: Make the column dense by using [Convert to Dataset](#) first or do not convert the column to categorical.

EXCEPTION MESSAGES

Sparse columns cannot be converted to Categorical.

Error 0075

Exception occurs when an invalid binning function is used when quantizing a dataset.

This error in Azure Machine Learning occurs when you are trying to bin data using an unsupported method, or when the parameter combinations are invalid.

Resolution:

Error handling for this event was introduced in an earlier version of Azure Machine Learning that allowed more customization of binning methods. Currently all binning methods are based on a selection from a dropdown list, so technically it should no longer be possible to get this error.

If you get this error when using the [Group Data into Bins](#) module, consider reporting the issue in the [Azure Machine Learning forum](#), providing the data types, parameter settings, and the exact error message.

EXCEPTION MESSAGES

Invalid binning function used.

Error 0077

Exception occurs when unknown blob file writes mode passed.

This error in Azure Machine Learning occurs if an invalid argument is passed in the specifications for a blob file destination or source.

Resolution: In almost all modules that import or export data to and from Azure blob storage, parameter values controlling the write mode are assigned by using a dropdown list; therefore, it is not possible to pass an invalid value, and this error should not appear. This error will be deprecated in a later release.

EXCEPTION MESSAGES

Unsupported blob writes mode.

Unsupported blob writes mode: {0}.

Error 0078

Exception occurs when the HTTP option for [Import Data](#) receives a 3xx status code indicating redirection.

This error in Azure Machine Learning occurs when the HTTP option for [Import Data](#) receives a 3xx (301, 302, 304, etc.) status code indicating redirection. You will receive this error if you attempt to connect to an HTTP source that redirects the browser to another page. For security reasons, redirecting websites are not allowed as data sources for Azure Machine Learning.

Resolution: If the website is a trusted website, enter the redirected URL directly.

EXCEPTION MESSAGES

Http redirection not allowed

Error 0079

Exception occurs if Azure storage container name is specified incorrectly.

This error in Azure Machine Learning occurs if the Azure storage container name is specified incorrectly. You will receive this error if you have not specified both the container and the blob (file) name using **the Path to blob beginning with container** option when writing to Azure Blob Storage.

Resolution: Revisit the [Export Data](#) module and verify that the specified path to the blob contains both the container and the file name, in the format **container/filename**.

EXCEPTION MESSAGES

The Azure storage container name is incorrect.

The Azure storage container name "{0}" is incorrect; a container name of the format container/blob was expected.

Error 0080

Exception occurs when column with all values missing is not allowed by module.

This error in Azure Machine Learning is produced when one or more of the columns consumed by the module contains all missing values. For example, if a module is computing aggregate statistics for each column, it cannot operate on a column containing no data. In such cases, module execution is halted with this exception.

Resolution: Revisit the input dataset and remove any columns that contain all missing values.

EXCEPTION MESSAGES

Columns with all values missing are not allowed.

Column {0} has all values missing.

Error 0081

Exception occurs in PCA module if number of dimensions to reduce to is equal to number of feature columns in input dataset, containing at least one sparse feature column.

This error in Azure Machine Learning is produced if the following conditions are met: (a) the input dataset has at least one sparse column and (b) the final number of dimensions requested is the same as the number of input dimensions.

Resolution: Consider reducing the number of dimensions in the output to be fewer than the number of dimensions in the input. This is typical in applications of PCA. For more information, see [Principal Component Analysis](#).

EXCEPTION MESSAGES

For dataset containing sparse feature columns number of dimensions to reduce should be less than number of feature columns.

Error 0082

Exception occurs when a model cannot be successfully serialized.

This error in Azure Machine Learning occurs when a saved machine learning model or transform cannot be loaded by a newer version of the Azure Machine Learning runtime as a result of a breaking change.

Resolution: The training experiment that produced the model or transform must be rerun and the model or transform must be resaved.

EXCEPTION MESSAGES

Model could not be serialized because it is likely serialized with an older serialization format. Retrain and resave the model.

Error 0083

Exception occurs if dataset used for training cannot be used for concrete type of learner.

This error in Azure Machine Learning is produced when the dataset is incompatible with the learner being trained. For example, the dataset might contain at least one missing value in each row, and as a result, the entire dataset would be skipped during training. In other cases, some machine learning algorithms such as anomaly detection do not expect labels to be present and can throw this exception if labels are present in the dataset.

Resolution: Consult the documentation of the learner being used to check requirements for the input dataset. Examine the columns to see all required columns are present.

EXCEPTION MESSAGES

Dataset used for training is invalid.

{0} contains invalid data for training.

{0} contains invalid data for training. Learner type: {1}.

Error 0084

Exception occurs when scores produced from an R Script are evaluated. This is currently unsupported.

This error in Azure Machine Learning occurs if you try to use one of the modules for evaluating a model with output from an R script that contains scores.

Resolution:

EXCEPTION MESSAGES

Evaluating scores produced by R is currently unsupported.

Error 0085

Exception occurs when script evaluation fails with an error.

This error in Azure Machine Learning occurs when you are running custom script that contains syntax errors.

Resolution: Review your code in an external editor and check for errors.

EXCEPTION MESSAGES

Error during evaluation of script.

The following error occurred during script evaluation, view the output log for more information: ----- Start of error message from {0} interpreter ----- {1} ----- End of error message from {0} interpreter -----

Error 0086

Exception occurs when a counting transform is invalid.

This error in Azure Machine Learning occurs when you select a transformation based on a count table, but the selected transform is incompatible with the current data, or with the new count table.

Resolution: The module supports saving the counts and rules that make up the transformation in two different formats. If you are merging count tables, verify that both tables you intend to merge use the same format.

In general, a count-based transform can only be applied to datasets that have the same schema as the dataset on which the transform was originally created.

For general information, see [Learning with Counts](#). For requirements specific to creating and merging count-based features, see these topics:

- [Merge Count Transform](#)
- [Import Count Table](#)
- [Modify Count Table Parameters](#)

EXCEPTION MESSAGES

Invalid counting transform specified.

The counting transform at input port '{0}' is invalid.

The counting transform at input port '{0}' cannot be merged with the counting transform at input port '{1}'. Check to verify the metadata used for counting matches.

Error 0087

Exception occurs when an invalid count table type is specified for learning with counts modules.

This error in Azure Machine Learning occurs when you try to import an existing count table, but the table is incompatible with the current data, or with the new count table.

Resolution: There are different formats for saving the counts and rules that make up the transformation. If you are merging count tables, verify that both use the same format.

Generally, a count-based transform can only be applied to datasets that have the same schema as the dataset on which the transform was originally created.

For general information, see [Learning with Counts](#). For requirements specific to creating and merging count-based features, see these topics:

Error 0088

Exception occurs when an invalid counting type is specified for learning with counts modules.

This error in Azure Machine Learning occurs when you try to use a different counting method than is supported for count-based featurization.

Resolution: In general, counting methods are chosen from a dropdown list, so you should not see this error.

For general information, see [Learning with Counts](#). For requirements specific to creating and merging count-based features, see these topics:

- [Merge Count Transform](#)
- [Import Count Table](#)
- [Modify Count Table Parameters](#)

EXCEPTION MESSAGES

Invalid counting type is specified.

The specified counting type '{0}' is not a valid counting type.

Error 0089

Exception occurs when the specified number of classes is less than the actual number of classes in a dataset used for counting.

This error in Azure Machine Learning occurs when you are creating a count table and the label column contains a different number of classes than you specified in the module parameters.

Resolution: Check your dataset and find out exactly how many distinct values (possible classes) there are in the label column. When you create the count table, you must specify at least this number of classes.

The count table cannot automatically determine the number of classes available.

When you create the count table, you cannot specify 0 or any number that is less than the actual number of classes in the label column.

EXCEPTION MESSAGES

The number of classes is incorrect. Make sure that the number of classes you specify in the parameter pane is greater than or equal to the number of classes in the label column.

The number of classes specified is '{0}', which is not greater than a label value '{1}' in the data set used to count. Make sure that the number of classes you specify in the parameter pane is greater than or equal to the number of classes in the label column.

Error 0090

Exception occurs when Hive table creation fails.

This error in Azure Machine Learning occurs when you are using [Export Data](#) or another option to save data to an HDInsight cluster and the specified Hive table cannot be created.

Resolution: Check the Azure storage account name associated with the cluster and verify that you are using the same account in the module properties.

EXCEPTION MESSAGES

The Hive table could not be created. For a HDInsight cluster, ensure the Azure storage account name associated with cluster is the same as what is passed in through the module parameter.

The Hive table "{0}" could not be created. For a HDInsight cluster, ensure the Azure storage account name associated with cluster is the same as what is passed in through the module parameter.

The Hive table "{0}" could not be created. For a HDInsight cluster, ensure the Azure storage account name associated with cluster is "{1}".

Error 0100

Exception occurs when an unsupported language is specified for a custom module.

This error in Azure Machine Learning occurs when building a custom module and the name property of the **Language** element in a custom module xml definition file has an invalid value. Currently, the only valid value for this property is `R`. For example:

```
<Language name="R" sourceFile="CustomAddRows.R" entryPoint="CustomAddRows" />
```

Resolution: Verify that the name property of the **Language** element in the custom module xml definition file is set to `R`. Save the file, update the custom module zip package, and try to add the custom module again.

EXCEPTION MESSAGES

Unsupported custom module language specified

Error 0101

All port and parameter IDs must be unique.

This error in Azure Machine Learning occurs when one or more ports or parameters are assigned the same ID value in a custom module XML definition file.

Resolution: Check that the ID values across all ports and parameters are unique. Save the xml file, update the custom module zip package, and try to add the custom module again.

EXCEPTION MESSAGES

All port and parameter IDs for a module must be unique

Module '{0}' has duplicate port/argument IDs. All port/argument IDs must be unique for a module.

Error 0102

Thrown when a ZIP file cannot be extracted.

This error in Azure Machine Learning occurs when you are importing a zipped package with the .zip extension, but the package is either not a zip file, or the file does not use a supported zip format.

Resolution: Make sure the selected file is a valid .zip file, and that it was compressed by using one of the supported compression algorithms.

If you get this error when importing datasets in compressed format, verify that all contained files use one of the supported file formats, and are in Unicode format. For more information, see [Unpack Zipped Datasets](#).

Try readding the desired files to a new compressed zipped folder and try to add the custom module again.

EXCEPTION MESSAGES

Given ZIP file is not in the correct format

Error 0103

Thrown when a ZIP file does not contain any .xml files

This error in Azure Machine Learning occurs when the custom module zip package does not contain any module definition (.xml) files. These files need to reside in the root of the zip package (for example, not within a subfolder.)

Resolution: Verify that one or more xml module definition files are in the root folder of the zip package by extracting it to a temporary folder on your disk drive. Any xml files should be directly in the folder you extracted the zip package to. Make sure when you create the zip package that you do not select a folder that contains xml files to zip as this will create a sub folder within the zip package with the same name as the folder you selected to zip.

EXCEPTION MESSAGES

Given ZIP file does not contain any module definition files (.xml files)

Error 0104

Thrown when a module definition file references a script that cannot be located

This error in Azure Machine Learning is thrown when a custom module xml definition file references a script file in the **Language** element that does not exist in the zip package. The script file path is defined in the **sourceFile** property of the **Language** element. The path to the source file is relative to the root of the zip package (same location as the module xml definition files). If the script file is in a sub folder, the relative path to the script file must be specified. For instance, if all scripts were stored in a **myScripts** folder within the zip package, the **Language** element would have to add this path to the **sourceFile** property as below. For example:

```
<Language name="R" sourceFile="myScripts/CustomAddRows.R" entryPoint="CustomAddRows" />
```

Resolution: Make sure that the value of the `sourceFile` property in the `Language` element of the custom module xml definition is correct and that the source file exists in the correct relative path in the zip package.

EXCEPTION MESSAGES

Referenced R script file does not exist.

Referenced R script file '{0}' cannot be found. Ensure that the relative path to the file is correct from the definitions location.

Error 0105

This error is displayed when a module definition file contains an unsupported parameter type

This error in Azure Machine Learning is produced when the you create a custom module xml definition and the type of a parameter or argument in the definition does not match a supported type.

Resolution: Make sure that the type property of any `Arg` element in the custom module xml definition file is a supported type.

EXCEPTION MESSAGES

Unsupported parameter type.

Unsupported parameter type '{0}' specified.

Error 0106

Thrown when a module definition file defines an unsupported input type

This error in Azure Machine Learning is produced when the type of an input port in a custom module XML definition does not match a supported type.

Resolution: Make sure that the type property of an `Input` element in the custom module XML definition file is a supported type.

EXCEPTION MESSAGES

Unsupported input type.

Unsupported input type '{0}' specified.

Error 0107

Thrown when a module definition file defines an unsupported output type

This error in Azure Machine Learning is produced when the type of an output port in a custom module xml definition does not match a supported type.

Resolution: Make sure that the type property of an `Output` element in the custom module xml definition file is a supported type.

EXCEPTION MESSAGES

Unsupported output type.

Unsupported output type '{0}' specified.

Error 0108

Thrown when a module definition file defines more input or output ports than are supported

This error in Azure Machine Learning is produced when too many input or output ports are defined in a custom module xml definition.

Resolution: Makes sure the maximum number of input and output ports defined in the custom module xml definition does not exceed the maximum number of supported ports.

EXCEPTION MESSAGES

Exceeded supported number of input or output ports.

Exceeded number of supported '{0}' ports. Maximum allowed number of '{0}' ports is '{1}'.

Error 0109

Thrown when a module definition file defines a column picker incorrectly

This error in Azure Machine Learning is produced when the syntax for a column picker argument contains an error in a custom module xml definition.

Resolution: This error is produced when the syntax for a column picker argument contains an error in a custom module xml definition.

EXCEPTION MESSAGES

Unsupported syntax for column picker.

Error 0110

Thrown when a module definition file defines a column picker that references a non-existent input port ID

This error in Azure Machine Learning is produced when the *portId* property within the Properties element of an Arg of type ColumnPicker does not match the ID value of an input port.

Resolution: Make sure the portId property matches the ID value of an input port defined in the custom module xml definition.

EXCEPTION MESSAGES

Column picker references a non-existent input port ID.

Column picker references a non-existent input port ID '{0}'.

Error 0111

Thrown when a module definition file defines an invalid property

This error in Azure Machine Learning is produced when an invalid property is assigned to an element in the custom module XML definition.

Resolution: Make sure the property is supported by the custom module element.

EXCEPTION MESSAGES

Property definition is invalid.

EXCEPTION MESSAGES

Property definition '{0}' is invalid.

Error 0112

Thrown when a module definition file cannot be parsed

This error in Azure Machine Learning is produced when there is an error in the xml format that prevents the custom module XML definition from being parsed as a valid XML file.

Resolution: Ensure that each element is opened and closed correctly. Make sure that there are no errors in the XML formatting.

EXCEPTION MESSAGES

Unable to parse module definition file.

Unable to parse module definition file '{0}'.

Error 0113

Thrown when a module definition file contains errors.

This error in Azure Machine Learning is produced when the custom module XML definition file can be parsed but contains errors, such as definition of elements not supported by custom modules.

Resolution: Make sure the custom module definition file defines elements and properties that are supported by custom modules.

EXCEPTION MESSAGES

Module definition file contains errors.

Module definition file '{0}' contains errors.

Module definition file '{0}' contains errors. {1}

Error 0114

Thrown when building a custom module fails.

This error in Azure Machine Learning is produced when a custom module build fails. This occurs when one or more custom module-related errors are encountered while adding the custom module. The additional errors are reported within this error message.

Resolution: Resolve the errors reported within this exception message.

EXCEPTION MESSAGES

Failed to build custom module.

Custom module builds failed with error(s): {0}

Error 0115

Thrown when a custom module default script has an unsupported extension.

This error in Azure Machine Learning occurs when you provide a script for a custom module that uses an unknown filename extension.

Resolution: Verify the file format and filename extension of any script files included in the custom module.

EXCEPTION MESSAGES

Unsupported extention for default script.

Unsupported file extention {0} for default script.

Error 0121

Thrown when SQL writes fails because the table is unwriteable

This error in Azure Machine Learning is produced when you are using the [Export Data](#) module to save results to a table in a SQL database, and the table cannot be written to. Typically, you will see this error if the [Export Data](#) module successfully establishes a connection with the SQL Server instance, but is then unable to write the contents of the Azure ML dataset to the table.

Resolution:

- Open the Properties pane of the [Export Data](#) module and verify that the database and table names are entered correctly.
- Review the schema of the dataset you are exporting, and make sure that the data is compatible with the destination table.
- Verify that the SQL sign in associated with the user name and password has permissions to write to the table.
- If the exception contains additional error information from SQL Server, use that information to make corrections.

EXCEPTION MESSAGES

Connected to server, unable to write to table.

Unable to write to Sql table: {0}

Error 0122

Exception occurs if multiple weight columns are specified and just one is allowed.

This error in Azure Machine Learning occurs when too many columns have been selected as weight columns.

Resolution: Review the input dataset and its metadata. Ensure that only one column contains weights.

EXCEPTION MESSAGES

Multiple weight columns are specified.

Error 0123

Exception occurs if column of vectors is specified to Label column.

This error in Azure Machine Learning occurs if you use a vector as the label column.

Resolution: Change the data format of the column if necessary, or choose a different column.

EXCEPTION MESSAGES

Column of vectors is specified as Label column.

Error 0124

Exception occurs if non-numeric columns are specified to be the weight column.

Resolution:

EXCEPTION MESSAGES

Non-numeric column is specified as the weight column.

Error 0125

Thrown when schema for multiple datasets does not match.

Resolution:

EXCEPTION MESSAGES

Dataset schema does not match.

Error 0126

Exception occurs if the user specifies a SQL domain that is not supported in Azure ML.

This error is produced when the user specifies a SQL domain that is not supported in Azure Machine Learning. You will receive this error if you are attempting to connect to a database server in a domain that is not on the allowed list. Currently, the allowed SQL domains are: ".database.windows.net", ".cloudapp.net", or ".database.secure.windows.net". That is, the server must be an Azure SQL server or a server in a virtual machine on Azure.

Resolution: Revisit the module. Verify that the SQL database server belongs to one of the accepted domains:

- .database.windows.net
- .cloudapp.net
- .database.secure.windows.net

EXCEPTION MESSAGES

Unsupported SQL domain.

The SQL domain {0} is not currently supported in Azure ML

Error 0127

Image pixel size exceeds allowed limit

This error occurs if you are reading images from an image dataset for classification and the images are larger than the model can handle.

Resolution: For more information about the image size and other requirements, see these topics:

- [Import Images](#)

- [Pretrained Cascade Image Classification](#)

EXCEPTION MESSAGES

Image pixel size exceeds allowed limit.

Image pixel size in the file '{0}' exceeds allowed limit: '{1}'

Error 0128

Number of conditional probabilities for categorical columns exceeds limit.

Resolution:

EXCEPTION MESSAGES

Number of conditional probabilities for categorical columns exceeds limit.

Number of conditional probabilities for categorical columns exceeds limit. Columns '{0}' and '{1}' are the problematic pair.

Error 0129

Number of columns in the dataset exceeds allowed limit.

Resolution:

EXCEPTION MESSAGES

Number of columns in the dataset exceeds allowed limit.

Number of columns in the dataset in '{0}' exceeds allowed.'

Number of columns in the dataset in '{0}' exceeds allowed limit of '{1}'.'

Number of columns in the dataset in '{0}' exceeds allowed '{1}' limit of '{2}'.'

Error 0130

Exception occurs when all rows in the training dataset contain missing values.

This occurs when some column in the training dataset is empty.

Resolution: Use the [Clean Missing Data](#) module to remove columns with all missing values.

EXCEPTION MESSAGES

All rows in training dataset contain missing values. Consider using the Clean Missing Data module to remove missing values.

Error 0131

Exception occurs if one or more datasets in a zip file fails to be unzipped and registered correctly

This error is produced when one or more datasets in a zip file fails to be unzipped and read correctly. You will receive this error if the unpacking fails because the zip file itself or one of the files in it is corrupt, or there is a system error while trying to unpack and expand a file.

Resolution: Use the details provided in the error message to determine how to proceed.

EXCEPTION MESSAGES

Upload zipped datasets failed

Zipped dataset {0} failed with the following message: {1}

Zipped dataset {0} failed with a {1} exception with message: {2}

Error 0132

No file name was specified for unpacking; multiple files were found in zip file.

This error is produced when no file name was specified for unpacking; multiple files were found in zip file. You will receive this error if the .zip file contains more than one compressed file, but you did not specify a file for extraction in the **Dataset to Unpack** text box, in the **Property** pane of the module. Currently, only one file can be extracted each time the module is run.

Resolution: The error message provides a list of the files found in the .zip file. Copy the name of the desired file and paste it into the **Dataset to Unpack** text box.

EXCEPTION MESSAGES

Zip file contains multiple files; you must specify the file to expand.

The file contains more than one file. Specify the file to expand. The following files were found: {0}

Error 0133

The specified file was not found in the zip file

This error is produced when the filename entered in the **Dataset to Unpack** field of the **Property** pane does not match the name of any file found in the .zip file. The most common causes of this error are a typing error or searching the wrong archive file for the file to expand.

Resolution: Revisit the module. If the name of the file you intended to decompress appears in the list of files found, copy the file name and paste it into the **Dataset to Unpack** property box. If you do not see the desired file name in the list, verify that you have the correct .zip file and the correct name for the desired file.

EXCEPTION MESSAGES

The specified file was not found int the zip file.

The specified file was not found. Found the following file(s): {0}

Error 0134

Exception occurs when label column is missing or has insufficient number of labeled rows.

This error occurs when the module requires a label column, but you did not include one in the column selection, or the label column is missing too many values.

This error can also occur when a previous operation changes the dataset such that insufficient rows are available to a downstream operation. For example, suppose you use an expression in the **Partition and Sample** module to divide a dataset by values. If no matches are found for your expression, one of the datasets resulting from the partition would be empty.

Resolution:

If you include a label column in the column selection but it isn't recognized, use the [Edit Metadata](#) module to mark it as a label column.

Use the [Summarize Data](#) module to generate a report that shows how many values are missing in each column. Then, you can use the [Clean Missing Data](#) module to remove rows with missing values in the label column.

Check your input datasets to make sure that they contain valid data, and enough rows to satisfy the requirements of the operation. Many algorithms will generate an error message if they require some minimum number rows of data, but the data contains only a few rows, or only a header.

EXCEPTION MESSAGES

Exception occurs when label column is missing or has insufficient number of labeled rows.

Exception occurs when label column is missing or has less than {0} labeled rows

Error 0135

Only centroid-based cluster is supported.

Resolution: You might encounter this error message if you have attempted to evaluate a clustering model that is based on a custom clustering algorithm that does not use centroids to initialize the cluster.

You can use [Evaluate Model](#) to evaluate clustering models that are based on the [K-Means Clustering](#) module. For custom algorithms, use the [Execute R Script](#) module to create a custom evaluation script.

EXCEPTION MESSAGES

Only centroid-based cluster is supported.

Error 0136

No file name was returned; unable to process the file as a result.

Resolution:

EXCEPTION MESSAGES

No file name was returned; unable to process the file as a result.

Error 0137

Azure Storage SDK encountered an error converting between table properties and dataset columns during read or write.

Resolution:

EXCEPTION MESSAGES

Conversion error between Azure table storage property and dataset column.

Conversion error between Azure table storage property and dataset column. Additional information: {0}

Error 0138

Memory has been exhausted, unable to complete running of module. Downsampling the dataset may help to alleviate the problem.

This error occurs when the module that is running requires more memory than is available in the Azure container. This can happen if you are working with a large dataset and the current operation cannot fit into memory.

Resolution: If you are trying to read a large dataset and the operation cannot be completed, downsampling the dataset might help.

If you use the visualizations on datasets to check the cardinality of columns, only some rows are sampled. To get a full report, use [Summarize Data](#). You can also use the [Apply SQL Transformation](#) to check for the number of unique values in each column.

Sometimes transient loads can lead to this error. Machine support also changes over time. See the [Azure Machine Learning FAQ](#) for a description of supported data size.

Try using [Principal Component Analysis](#) or one of the provided feature selection methods to reduce your dataset to a smaller set of more feature-rich columns: [Feature Selection](#)

EXCEPTION MESSAGES

Memory has been exhausted, unable to complete running of module.

Error 0139

Exception occurs when it is not possible to convert a column to another type.

This error in Azure Machine Learning occurs when you try to convert a column to a different data type, but that type is not supported by the current operation or by the module.

The error might also appear when a module tries to implicitly convert data to meet requirements of the current module, but the conversion is not possible.

Resolution:

1. Review your input data and determine the exact data type of the column that you want to use, and the data type of the column that is producing the error. Sometimes you might think the data type is correct, but find that an upstream operation has modified the data type or usage of a column. Use the [Edit Metadata](#) module to reset column metadata to its original state.
2. Look at the module help page to verify the requirements for the specified operation. Determine which data types are supported by the current module, and what range of values is supported.
3. If values need to be truncated, rounded, or outliers removed, use the [Apply Math Operation](#) or [Clip Values](#) modules to make corrections.
4. Consider whether it is possible to convert or cast the column to a different data type. The following modules all provide considerable flexibility and power for modifying data:
 - [Apply SQL Transformation](#)
 - [Execute R Script](#)
 - [Execute Python Script](#).

NOTE

Still not working? Consider providing additional feedback on the problem, to help us develop better troubleshooting guidance. Just submit feedback on this page and provide the name of the module that generated the error, and the data type conversion that failed.

EXCEPTION MESSAGES

Not allowed conversion.

Could not convert: {0}.

Could not convert: {0}, on row {1}.

Could not convert column of type {0} to column of type {1} on row {2}.

Could not convert column "{2}" of type {0} to column of type {1} on row {3}.

Could not convert column "{2}" of type {0} to column "{3}" of type {1} on row {4}.

Error 0140

Exception occurs if passed column set argument does not contain other columns except label column.

This error occurs if you connected a dataset to a module that requires multiple columns, including features, but you have provided only the label column.

Resolution: Choose at least one feature column to include in the dataset.

EXCEPTION MESSAGES

Specified column set does not contain other columns except label column.

Error 0141

Exception occurs if the number of the selected numerical columns and unique values in the categorical and string columns is too small.

This error in Azure Machine Learning occurs when there are not enough unique values in the selected column to perform the operation.

Resolution: Some operations perform statistical operations on feature and categorical columns, and if there are not enough values, the operation might fail or return an invalid result. Check your dataset to see how many values there are in the feature and label columns, and determine whether the operation you are trying to perform is statistically valid.

If the source dataset is valid, you might also check whether some upstream data manipulation or metadata operation has changed the data and removed some values.

If upstream operations include splitting, sampling, or resampling, verify that the outputs contain the expected number of rows and values.

EXCEPTION MESSAGES

The number of the selected numerical columns and unique values in the categorical and string columns is too small.

The total number of the selected numerical columns and unique values in the categorical and string columns (currently {0}) should be at least {1}

Error 0142

Exception occurs when the system cannot load certificate to authenticate.

Resolution:

EXCEPTION MESSAGES

The certificate cannot be loaded.

The certificate {0} cannot be loaded. Its thumbprint is {1}.

Error 0143

Can't parse user-provided URL that is supposed to be from GitHub.

This error in Azure Machine Learning occurs when you specify an invalid URL and the module requires a valid GitHub URL.

Resolution: Verify that the URL refers to a valid GitHub repository. Other site types are not supported.

EXCEPTION MESSAGES

URL is not from github.com.

URL is not from github.com: {0}

Error 0144

User-provided GitHub url is missing the expected part.

This error in Azure Machine Learning occurs when you specify a GitHub file source using an invalid URL format.

Resolution: Check that the URL of the GitHub repository is valid and ends with \blob\ or \tree\.

EXCEPTION MESSAGES

Cannot parse GitHub URL.

Cannot parse GitHub URL (expecting '\blob\' or '\tree\' after the repository name): {0}

Error 0145

Cannot create the replication directory for some reason.

This error in Azure Machine Learning occurs when the module fails to create the specified directory.

Resolution:

EXCEPTION MESSAGES

Cannot create replication directory.

Error 0146

When the user files are unzipped into the local directory, the combined path might be too long.

This error in Azure Machine Learning occurs when you are extracting files but some file names are too long when unzipped.

Resolution: Edit the file names such that combined path and file name is no longer than 248 characters.

EXCEPTION MESSAGES

Replication path is longer than 248 characters, shorten the script name or path.

Error 0147

Could not download stuff from GitHub for some reason

This error in Azure Machine Learning occurs when you cannot read or download the specified files from GitHub.

Resolution: The issue might be temporary; you might try accessing the files at another time. Or verify that you have the necessary permissions and that the source is valid.

EXCEPTION MESSAGES

GitHub access error.

GitHub access error. {0}

Error 0148

Unauthorized access issues while extracting data or creating directory.

This error in Azure Machine Learning occurs when you are trying to create a directory or read data from storage but do not have the necessary permissions.

Resolution:

EXCEPTION MESSAGES

Unauthorized access exception while extracting data.

Error 0149

The user file does not exist inside GitHub bundle.

This error in Azure Machine Learning occurs when the specified file cannot be found.

Resolution:

EXCEPTION MESSAGES

GitHub file is not found.

GitHub file is not found.: {0}

Error 0150

The scripts that come from the user package could not be unzipped, most likely because of a collision with GitHub files.

This error in Azure Machine Learning occurs when a script cannot be extracted, typically when there is an existing file of the same name.

Resolution:

EXCEPTION MESSAGES

Unable to unzip the bundle; possible name collision with GitHub files.

Error 0151

There was an error writing to cloud storage. Check the URL.

This error in Azure Machine Learning occurs when the module tries to write data to cloud storage but the URL is unavailable or invalid.

Resolution: Check the URL and verify that it is writable.

EXCEPTION MESSAGES

Error writing to cloud storage (possibly a bad url).

Error writing to cloud storage: {0}. Check the url.

Error 0152

The Azure cloud type was specified incorrectly in the module context.

EXCEPTION MESSAGES

Bad Azure Cloud Type

Bad Azure Cloud Type: {0}

Error 0153

The storage end point specified is invalid.

EXCEPTION MESSAGES

Bad Azure Cloud Type

Bad Storage End Point: {0}

Error 0154

The specified server name could not be resolved

EXCEPTION MESSAGES

The specified server name could not be resolved

The specified server {0}.documents.azure.com could not be resolved

Error 0155

The DocDb Client threw an exception

EXCEPTION MESSAGES

The DocDb Client threw an exception

DocDb Client: {0}

Error 0156

Bad response for HCatalog Server.

EXCEPTION MESSAGES

Bad response for HCatalog Server. Check that all services are running.

Bad response for HCatalog Server. Check that all services are running. Error details: {0}

Error 0157

There was an error reading from Azure Cosmos DB due to inconsistent or different document schemas. Reader requires all documents to have the same schema.

EXCEPTION MESSAGES

Detected documents with different schemas. Make sure all documents have the same schema

Error 1000

Internal library exception.

This error is provided to capture otherwise unhandled internal engine errors. Therefore, the cause for this error might be different depending on the module that generated the error.

To get more help, we recommend that you post the detailed message that accompanies the error to the Azure Machine Learning forum, together with a description of the scenario, including the data used as inputs. This feedback will help us to prioritize errors and identify the most important issues for further work.

EXCEPTION MESSAGES

Library exception.

Library exception: {0}

{0} library exception: {1}

More help

[Module error codes](#)

Need more help or troubleshooting tips for Azure Machine Learning? Try these resources:

- [Troubleshooting guide: Create and connect to an Machine Learning workspace](#)
- [Azure Machine Learning Frequently Asked Questions \(FAQ\)](#)