Contents

Algorithm & component reference for Azure Machine Learning designer

3/28/2022 • 3 minutes to read • Edit Online

This reference content provides the technical background on each of the machine learning algorithms and components available in Azure Machine Learning designer.

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

For help with choosing algorithms, see

- How to select algorithms
- Azure Machine Learning Algorithm Cheat Sheet

> **TIP**
> In any pipeline in the designer, you can get information about a specific component. Select the **Learn more** link in the component card when hovering on the component in the component list, or in the right pane of the component.

Data preparation components

FUNCTIONALITY	DESCRIPTION	COMPONENT
Data Input and Output	Move data from cloud sources into your pipeline. Write your results or intermediate data to Azure Storage, or SQL Database, while running a pipeline, or use cloud storage to exchange data between pipelines.	Enter Data Manually Export Data Import Data
Data Transformation	Operations on data that are unique to machine learning, such as normalizing or binning data, dimensionality reduction, and converting data among various file formats.	Add Columns Add Rows Apply Math Operation Apply SQL Transformation Clean Missing Data Clip Values Convert to CSV Convert to Dataset Convert to Indicator Values Edit Metadata Group Data into Bins Join Data Normalize Data Partition and Sample Remove Duplicate Rows SMOTE Select Columns Transform Select Columns in Dataset Split Data

FUNCTIONALITY	DESCRIPTION	COMPONENT
Feature Selection	Select a subset of relevant, useful features to use to build an analytical model.	Filter Based Feature Selection Permutation Feature Importance
Statistical Functions	Provide a wide variety of statistical methods related to data science.	Summarize Data

Machine learning algorithms

FUNCTIONALITY	DESCRIPTION	COMPONENT
Regression	Predict a value.	Boosted Decision Tree Regression Decision Forest Regression Fast Forest Quantile Regression Linear Regression Neural Network Regression Poisson Regression
Clustering	Group data together.	K-Means Clustering
Classification	Predict a class. Choose from binary (two-class) or multiclass algorithms.	Multiclass Boosted Decision Tree Multiclass Decision Forest Multiclass Logistic Regression Multiclass Neural Network One vs. All Multiclass One vs. One Multiclass Two-Class Averaged Perceptron Two-Class Boosted Decision Tree Two-Class Decision Forest Two-Class Logistic Regression Two-Class Neural Network Two Class Support Vector Machine

Components for building and evaluating models

FUNCTIONALITY	DESCRIPTION	COMPONENT
Model Training	Run data through the algorithm.	Train Clustering Model Train Model Train Pytorch Model Tune Model Hyperparameters
Model Scoring and Evaluation	Measure the accuracy of the trained model.	Apply Transformation Assign Data to Clusters Cross Validate Model Evaluate Model Score Image Model Score Model
Python Language	Write code and embed it in a component to integrate Python with your pipeline.	Create Python Model Execute Python Script

FUNCTIONALITY	DESCRIPTION	COMPONENT
R Language	Write code and embed it in a component to integrate R with your pipeline.	Execute R Script
Text Analytics	Provide specialized computational tools for working with both structured and unstructured text.	Convert Word to Vector Extract N Gram Features from Text Feature Hashing Preprocess Text Latent Dirichlet Allocation Score Vowpal Wabbit Model Train Vowpal Wabbit Model
Computer Vision	Image data preprocessing and Image recognition related components.	Apply Image Transformation Convert to Image Directory Init Image Transformation Split Image Directory DenseNet ResNet
Recommendation	Build recommendation models.	Evaluate Recommender Score SVD Recommender Score Wide and Deep Recommender Train SVD Recommender Train Wide and Deep Recommender
Anomaly Detection	Build anomaly detection models.	PCA-Based Anomaly Detection Train Anomaly Detection Model

Web service

Learn about the web service components, which are necessary for real-time inference in Azure Machine Learning designer.

Error messages

Learn about the error messages and exception codes that you might encounter using components in Azure Machine Learning designer.

Next steps

- Tutorial: Build a model in designer to predict auto prices

Import Data component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to load data into a machine learning pipeline from existing cloud data services.

> **NOTE**
> All functionality provided by this component can be done by **datastore** and **datasets** in the worksapce landing page. We recommend you use **datastore** and **dataset** which includes additional features like data monitoring. To learn more, see How to Access Data and How to Register Datasets article. After you register a dataset, you can find it in the **Datasets -> My Datasets** category in designer interface. This component is reserved for Studio(classic) users to for a familiar experience.

The **Import Data** component support read data from following sources:

- URL via HTTP
- Azure cloud storages through **Datastores**)
  - Azure Blob Container
  - Azure File Share
  - Azure Data Lake
  - Azure Data Lake Gen2
  - Azure SQL Database
  - Azure PostgreSQL

Before using cloud storage, you must register a datastore in your Azure Machine Learning workspace first. For more information, see How to Access Data.

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 designer pipeline. The output of **Import Data** is a dataset that can be used with any designer pipeline.

If your source data changes, you can refresh the dataset and add new data by rerunning Import Data.

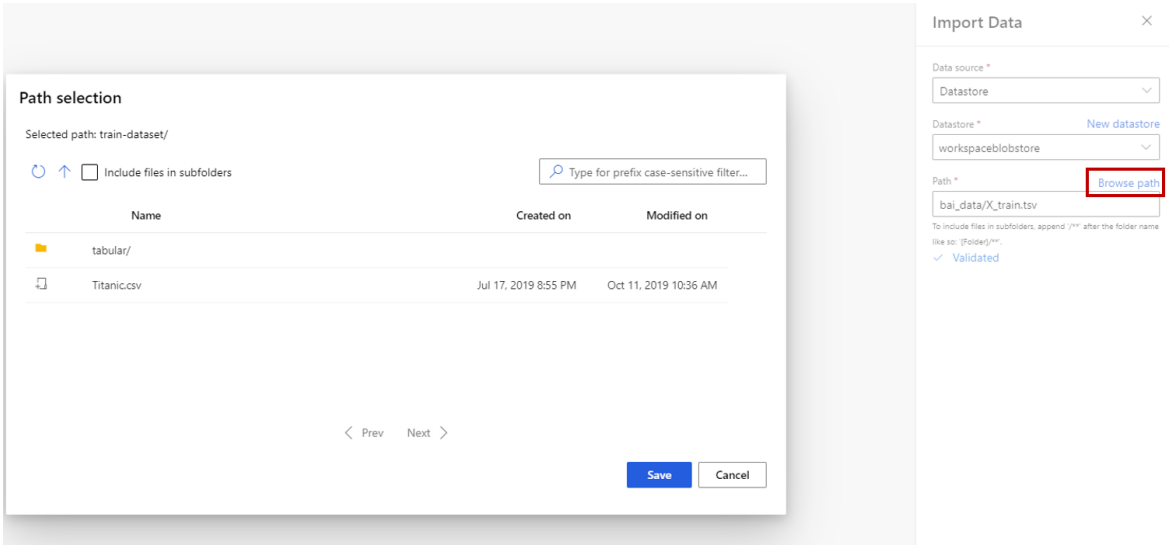
> **WARNING**
> If your workspace is in a virtual network, you must configure your datastores to use the designer's data visualization features. For more information on how to use datastores and datasets in a virtual network, see Use Azure Machine Learning studio in an Azure virtual network.

How to configure Import Data

1. Add the **Import Data** component to your pipeline. You can find this component in the **Data Input and Output** category in the designer.
2. Select the component to open the right pane.
3. Select **Data source**, and choose the data source type. It could be HTTP or datastore.

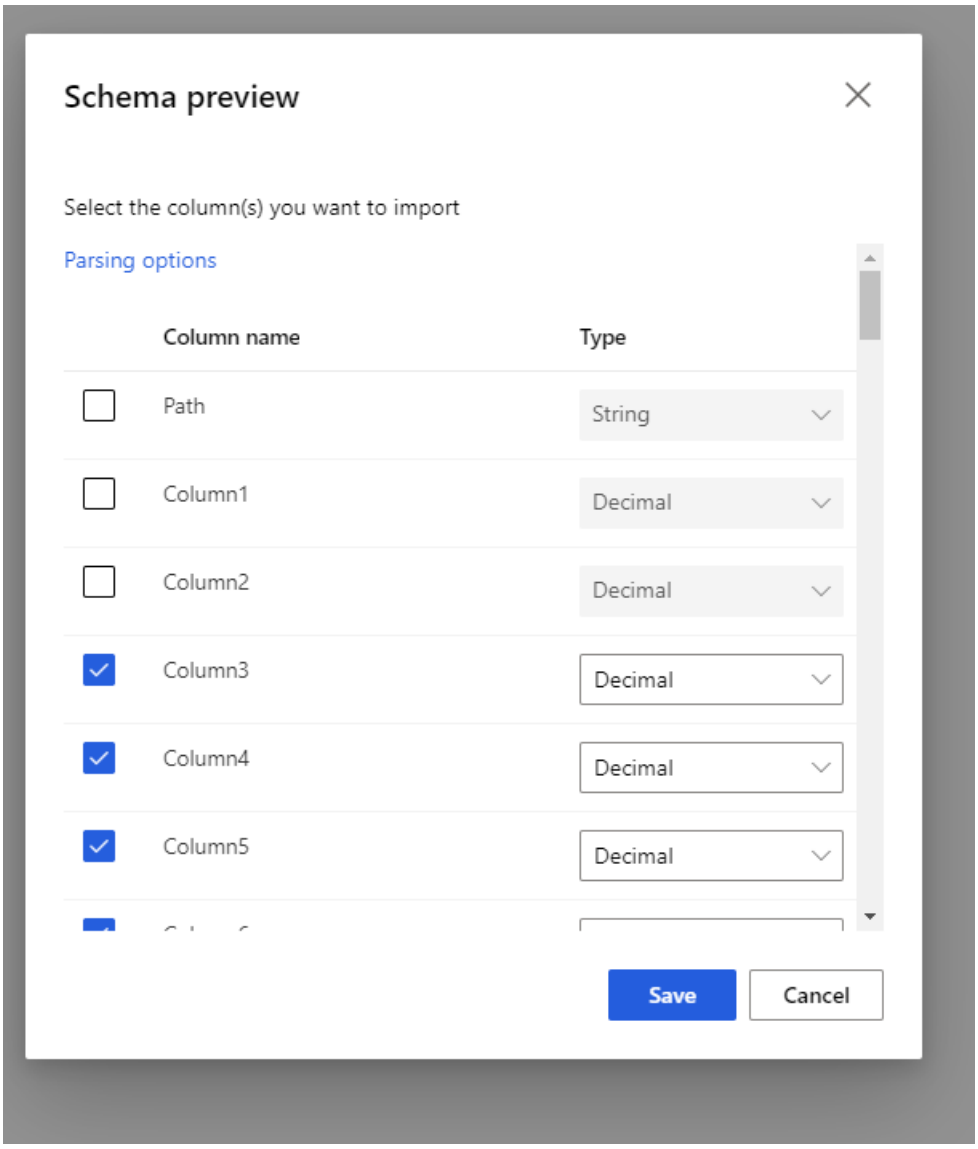
   If you choose datastore, you can select existing datastores that are already registered to your Azure

Machine Learning workspace or create a new datastore. Then define the path of data to import in the datastore. You can easily browse the path by selecting **Browse Path**.



> **NOTE**
>
> **Import Data** component is for **Tabular** data only. If you want to import multiple tabular data files one time, it requires following conditions, otherwise errors will occur:
>
> 1. To include all data files in the folder, you need to input `folder_name/**` for **Path**.
> 2. All data files must be encoded in unicode-8.
> 3. All data files must have the same column numbers and column names.
> 4. The result of importing multiple data files is concatenating all rows from multiple files in order.

4. Select the preview schema to filter the columns you want to include. You can also define advanced settings like Delimiter in Parsing options.



5. The checkbox, **Regenerate output**, decides whether to execute the component to regenerate output at running time.

   It's by default unselected, which means if the component has been executed with the same parameters previously, the system reuses the output from last run to reduce run time.

   If it is selected, the system executes the component again to regenerate output. So select this option when underlying data in storage is updated, it can help to get the latest data.

6. Submit the pipeline.

   When Import Data loads the data into the designer, 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, right-click the output dataset and select **Visualize** to see if the data was imported successfully.

If you want to save the data for reuse, rather than importing a new set of data each time the pipeline is run, select the **Register dataset** icon under the **Outputs+logs** tab in the right panel of the component. Choose a

name for the dataset. The saved dataset preserves the data at the time of saving. The dataset is not updated when the pipeline is rerun, even if the dataset in the pipeline changes. This can be useful for taking snapshots of data.

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

- Use Edit Metadata to change column names, handle a column as a different data type, or 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 component.
- Use Partition and Sample to divide the dataset, perform sampling, or get the top n rows.

Limitations

Due to datastore access limitation, if your inference pipeline contains **Import Data** component, it is auto-removed when deployed to real-time endpoint.

Next steps

See the set of components available to Azure Machine Learning.

Enter Data Manually component

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use the **Enter Data Manually** component to create a small dataset by typing values. The dataset can have multiple columns.

This component can be helpful in scenarios such as:

- Generating a small set of values for testing.
- Creating a short list of labels.
- Typing a list of column names to insert in a dataset.

Create a dataset

1. Add the Enter Data Manually component to your pipeline. You can find this component in the **Data Input and Output** category in Azure Machine Learning.
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**: Attribute-relation file format used by Weka.
   - **CSV**: Comma-separated values format. For more information, see Convert to CSV.
   - **SVMLight**: Format used by Vowpal Wabbit and other machine learning frameworks.
   - **TSV**: Tab-separated values format.

   If you choose a format and do not provide data that meets the format specifications, a runtime 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 by 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 column names (Col1, Col2, and so forth) are used. You can add or change columns names later by using Edit Metadata.

   - **TSV**: To create multiple columns, paste in tab-separated text, or type multiple columns by 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 column names (Col1, Col2, and so forth) are used. You can add or change columns names later by using Edit Metadata.

   - **ARFF**: Paste in an existing ARFF format file. If you're 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 can be added to a simple list. The column heading would be `SampleText`. Note that the String type is not supported.

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

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

  For example, the following sample represents the first couple of 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 component, 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. Select the Enter key after each row, to start a new line.

   If you select Enter multiple times to add multiple empty trailing rows, the empty rows will be removed or trimmed.

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

5. Connect the output port to other components, and run the pipeline.

   To view the dataset, right-click the component and select **Visualize**.

Next steps

See the set of components available to Azure Machine Learning.

Export Data component

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to save results, intermediate data, and working data from your pipelines into cloud storage destinations.

This component supports exporting your data to the following cloud data services:

- Azure Blob Container
- Azure File Share
- Azure Data Lake Storage Gen1
- Azure Data Lake Storage Gen2
- Azure SQL database

Before exporting your data, you need to first register a datastore in your Azure Machine Learning workspace. For more information, see Access data in Azure storage services.

How to configure Export Data

1. Add the **Export Data** component to your pipeline in the designer. You can find this component in the **Input and Output** category.
2. Connect **Export Data** to the component that contains the data you want to export.
3. Select **Export Data** to open the **Properties** pane.
4. For **Datastore**, select an existing datastore from the dropdown list. You can also create a new datastore. Check how by visiting Access data in Azure storage services.

   > **NOTE**
   > Exporting data of a certain data type to a SQL database column specified as another data type is not supported. The target table does not need to exist first.

5. The checkbox, **Regenerate output**, decides whether to execute the component to regenerate output at running time.

   It's by default unselected, which means if the component has been executed with the same parameters previously, the system will reuse the output from last run to reduce run time.

   If it is selected, the system will execute the component again to regenerate output.

6. Define the path in the datastore where the data is. The path is a relative path.Take `data/testoutput` as an example, which means the input data of **Export Data** will be exported to `data/testoutput` in the datastore you set in the **Output settings** of the component.

   > **NOTE**
   > The empty paths or **URL paths** are not allowed.

7. For **File format**, select the format in which data should be stored.

8. Submit the pipeline.

Limitations

Due to datstore access limitation, if your inference pipeline contains **Export Data** component, it will be auto-removed when deploy to real-time endpoint.

Next steps

See the set of components available to Azure Machine Learning.

Add Columns component

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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**.

How to configure Add Columns

1. Add the **Add Columns** component to your pipeline.
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. Submit the pipeline.

Results

After the pipeline has run:

- To see the first rows of the new dataset, right-click the **Add Columns** component and select Visualize. Or Select the component and switch to the **Outputs** tab in the right panel, click on the histogram icon in the **Port outputs** to visualize the result.

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. For example, if there are two instances of a column named TargetOutcome, the left column would be renamed TargetOutcome_1 and the right column would be renamed TargetOutcome_2.

Next steps

See the set of components available to Azure Machine Learning.

Add Rows component

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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** component into your pipeline, You can find it under **Data Transformation**.
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. Submit the pipeline. 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** component, the dataset is duplicated.

Next steps

See the set of components available to Azure Machine Learning.

Apply Math Operation

3/28/2022 • 15 minutes to read • Edit Online

This article describes a component of Azure Machine Learning designer.

Use the Apply Math Operation to create calculations that are applied to numeric columns in the input dataset.

Math operations include arithmetic functions, trigonometric functions, rounding functions, and special functions used in data science such as gamma and error functions.

After you define an operation and run the pipeline, the values are added to your dataset. Depending on how you configure the component, you can:

- Append the results to your dataset (useful when verifying the result of the operation).
- Replace columns values with the new, computed values.
- Generate a new column for results, and not show the original data.

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

  The **Operations** category includes 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

  The **Rounding** 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

  The **Special** category includes mathematical functions that are especially used in data science, such as elliptic integrals and the Gaussian error function.

- Trigonometric

  The **Trigonometric** 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. Ensure that the input column is the appropriate type and

contains the correct type of values for the specified operation.

How to configure Apply Math Operation

The **Apply Math Operation** component 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 component 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** component.

1. Add the **Apply Math Operation** component to your pipeline.
2. Connect a dataset that contains at least one numeric column.
3. 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, Ln(-1.0) is an invalid operation and results in a value of `NaN`.
4. Select **Category** to select the **type** of math operation to perform.
5. Choose a specific operation from the list in that category.
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. Submit the pipeline.

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 **In place** option, the source data is not deleted or changed; the column in the original dataset is still available in the designer. To view the original data, you can connect the Add Columns component 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 or a column set 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 function 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.

In **Column set**, select the column that contains the exponent values t.

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 x as a constant, or a value in a column.

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.

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 values for the selected column.

Log2

Returns the base 2-logarithm values for 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 **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 contain 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 **Second argument** text box.

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 designer anytime that you need to test two sets of values against each other. For example, in a pipeline 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. 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 for division and subtraction are as follows:

- Subtract(Arg1_Arg2) = Arg1 - Arg 2
- Divide(Arg1_Arg2) = Arg1 / Arg 2

The following table shows some examples

OPERATION	NUM1	NUM2	RESULT COLUMN	RESULT VALUE
Addition	1	5	Add(Num2_Num1)	6
Multiplication	1	5	Multiple(Num2_Num 1)	5
Subtraction	5	1	Subtract(Num2_Num 1)	4
Subtraction	0	1	Subtract(Num2_Num 1)	-1
Division	5	1	Divide(Num2_Num1)	5
Division	1	0	Divide(Num2_Num1)	Infinity

Add

Specify the source columns by using **Column set**, and then add to those values a number specified in **Second argument**.

To add the values in two columns, choose a column or columns using **Column set**, and then choose a second column using **Second argument**.

Divide

Divides the values in **Column set** by a constant or by the column values defined in **Second 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 column of values to operate on (the *minuend*), by choosing a different column, using the **Column set** option. Then, specify the number to subtract (the *subtrahend*) by using the **Second argument** dropdown list. You can choose either a constant or column of values.

Rounding operations

Azure Machine Learning designer 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 component 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 – 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.

ExponentialIntegralEin

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.

GammaRegularizedPInverse

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 iIncludes 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.

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

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 components to do all the calculations at once, using SQL, R, or Python script:

- Execute R Script
- Execute Python Script
- Apply SQL Transformation

Next steps

See the set of components available to Azure Machine Learning.

Apply SQL Transformation

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component of Azure Machine Learning designer.

Using the Apply SQL Transformation component, 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 component is **SQLite**. For more information about SQLite syntax, see SQL as Understood by SQLite. This component will bump data to SQLite, which is in the memory DB, hence the component execution requires much more memory and may hit an `Out of memory` error. Make sure your computer has enough RAM.

How to configure Apply SQL Transformation

The component 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.

Following is sample code to show how to join two tables. t1 and t2 are two datasets connected to the left and middle input ports of **Apply SQL Transformation**:

```
SELECT t1.*
    , t3.Average_Rating
FROM t1 join
    (SELECT placeID
        , AVG(rating) AS Average_Rating
    FROM t2
    GROUP BY placeID
    ) as t3
on t1.placeID = t3.placeID
```

The remaining parameter is a SQL query, which uses the SQLite syntax. 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.

This component supports all standard statements of the SQLite syntax. For a list of unsupported statements, see the Technical Notes section.

Technical notes

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

- An input is always required on port 1.
- 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.
- If you have used **Edit Metadata** to specify the column metadata (categorical or fields) before **Apply**

SQL Transformation, the outputs of **Apply SQL Transformation** will not contain these attributes. You need to use **Edit Metadata** to edit the column after **Apply SQL Transformation**.

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

Next steps

See the set of components available to Azure Machine Learning.

Clean Missing Data component

3/28/2022 • 6 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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 component supports multiple types 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

Using this component 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 component 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 component.

How to use Clean Missing Data

This component 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 sections 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 component 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 component.

1. Add the Clean Missing Data component to your pipeline, 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. Select the **Clean Missing Data** component, and click on **Edit column** in the right panel of the

component.

b. For **Include**, select **Column types** from the dropdown list, and then select **Numeric**.

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 component returns an error and stops the pipeline.

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.

   > **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.

5. For **Cleaning Mode**, select one of the following options for replacing or removing missing values:

   - **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.
   - **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.
   - **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.

- **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.

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 String.

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. Submit the pipeline.

Results

The component 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 component to your pipeline.
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 the designer. Locate the saved transformation and drag it into the pipeline.
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 columns 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. Submit the pipeline.

Next steps

See the set of components available to Azure Machine Learning.

Clip Values

3/28/2022 • 5 minutes to read • Edit Online

This article describes a component of Azure Machine Learning designer.

Use the Clip Values component to identify and optionally replace data values that are above or below a specified threshold with a mean, a constant, or other substitute value.

You connect the component 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 component 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 component 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** component to your pipeline and connect it to the dataset you want to modify. You can find this component 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.
   - **ClipSubpeaks**: When you clip values by subpeaks, you specify only a lower boundary. Values that are less than that boundary value are replaced.
   - **ClipPeaksAndSubpeaks**: When you clip values by peaks and subpeaks, you can specify both the upper and lower boundaries. Values that are outside that range are replaced. 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 for 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 for lower threshold**, and type 80 for **Percentile value for 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 specified are considered to be inside the allowed range of values, and thus are not replaced. 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 useful 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. Submit the pipeline.

    Right-click the **Clip Values** component and select **Visualize** or select the component and switch to the **Outputs** tab in the right panel, click on the histogram icon in the **Port outputs**, to review the values and make sure the clipping operation met your expectations.

Examples for 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 pipeline 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 = 20
2. Replace with threshold; Upper percentile = 60; Lower percentile = 40

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

Next steps

See the set of components available to Azure Machine Learning.

Convert to CSV component

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to convert a dataset into a CSV format that can be downloaded, exported, or shared with R or Python script components.

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. 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, 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.

When you convert a dataset to CSV, the csv is saved in your Azure ML workspace. You can use an Azure storage utility to open and use the file directly. You can also access the CSV in the designer by selecting the **Convert to CSV** component, then select the histogram icon under the **Outputs** tab in the right panel to view the output. You can download the CSV from the Results folder to a local directory.

How to configure Convert to CSV

1. Add the Convert to CSV component to your pipeline. You can find this component in the **Data Transformation** group in the designer.
2. Connect it to any component that outputs a dataset.
3. Submit the pipeline.

Results

Select the **Outputs** tab in the right panel of **Convert to CSV**, and select on one of these icons under the **Port outputs**.

- **Register dataset**: Select the icon and save the CSV file back to the Azure ML workspace as a separate dataset. You can find the dataset as a component in the component tree under the **My Datasets** section.
- **View output**: Select the eye icon, and follow the instruction to browse the **Results_dataset** folder, and download the data.csv file.

Next steps

See the set of components available to Azure Machine Learning.

Convert to Dataset

3/28/2022 • 3 minutes to read • Edit Online

This article describes how to use the Convert to Dataset component in Azure Machine Learning designer to convert any data for a pipeline to the designer's internal format.

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

We recommend saving data to the dataset format if you've performed some kind of normalization or cleaning on a set of data, and you want to ensure that the changes are used in other pipelines.

> **NOTE**
> Convert to Dataset changes only the format of the data. 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 enter a new name.

How to use Convert to Dataset

We recommend that you use the Edit Metadata component to prepare the dataset before you use Convert to Dataset. You can add or change column names, adjust data types, and make other changes as needed.

1. Add the Convert to Dataset component to your pipeline. You can find this component in the **Data transformation** category in the designer.
2. Connect it to any component that outputs a dataset.

   As long as the data is tabular, you can convert it to a dataset. This includes data loaded through Import Data, data created through Enter Data Manually, or datasets transformed through Apply Transformation.
3. In the **Action** drop-down list, indicate if you want to do any cleanup on the data before you save the dataset:

   - **None**: Use the data as is.
   - **SetMissingValue**: Set a specific value to a missing value in the dataset. The default placeholder is the question mark character (?), but you can use the **Custom missing value** option to enter a different value. For example, if you enter **Taxi** for **Custom missing value**, then all instances of **Taxi** in the dataset will be changed to the missing value.
   - **ReplaceValues**: Use this option to specify a single exact value to be replaced with any other exact value. You can replace missing values or custom values by setting the **Replace** method:
     - **Missing**: Choose this option to replace missing values in the input dataset. For **New Value**, enter the value to replace the missing values with.
     - **Custom**: Choose this option to replace custom values in the input dataset. For **Custom value**, enter the value that you want to find. For example, if your data contains the string `obs` used as a placeholder for missing values, you enter `obs`. For **New value**, enter the new value to replace the original string with.

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

Results

- To save the resulting dataset with a new name, select on the icon **Register dataset** under the **Outputs** tab in the right panel of the component.

Technical notes

- Any component that takes a dataset as input can also take data in the CSV file or the TSV file. Before any component code is run, the inputs are preprocessed. Preprocessing is equivalent to running the Convert to Dataset component on the input.
- You can't convert from the SVMLight format to a dataset.
- When you're 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 can't 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.

Next steps

See the set of components available to Azure Machine Learning.

Convert to Indicator Values

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component of Azure Machine Learning designer.

Use the **Convert to Indicator Values** component in Azure Machine Learning designer to convert columns that contain categorical values into a series of binary indicator columns.

This component also outputs a definition of the transformation used to convert to indicator values. You can reuse this transformation on other datasets that have the same schema, by using the Apply Transformation component.

How to configure Convert to Indicator Values

1. Find the **Convert to Indicator Values** and drag it to your pipeline draft. You can find this component under **Data Transformation** category.

   > **NOTE**
   > You can use the Edit Metadata component before the **Convert to Indiciator Values** component to mark the target column(s) as categorical.

2. Connect the **Convert to Indicator Values** component to the dataset containing the columns you want to convert.

3. Select **Edit column** to choose one or more categorical columns.

4. Select the **Overwrite categorical columns** option if you want to output **only** the new Boolean columns. By default, this option is off.

   > **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 component at any time to add the source column back in.

5. Submit the pipeline.

Results

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

SERVER ID	FAILURE SCORE
10303	High

When you apply **Convert to Indicator Values**, the designer converts a single column of labels 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's 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. So, 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 since each server can have only one risk rating.

You can now use the three indicator columns as features in a machine learning model.

The component returns two outputs:

- **Results dataset**: A dataset with converted indicator values columns. Columns not selected for cleaning are also "passed through".
- **Indicator values transformation**: A data transformation used for converting to indicator values, that can be saved in your workspace and applied to new data later.

Apply a saved indicator values operation to new data

If you need to repeat indicator values operations often, you can save your data manipulation steps as a *transform* to reuse it with the same dataset. This is useful if you must frequently reimport and then clean data that have the same schema.

1. Add the Apply Transformation component to your pipeline.
2. Add the dataset you want to clean, and connect the dataset to the right-hand input port.
3. Expand the **Data Transformation** group in the left-hand pane of designer. Locate the saved transformation and drag it into the pipeline.
4. Connect the saved transformation to the left input port of Apply Transformation.

   When you apply a saved transformation, you cannot select which columns to transform. This is because the transformation has been defined and applies automatically to the data types specified in the original operation.
5. Submit the pipeline.

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 the following 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.

- 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 may 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
Contoso Cars	0	0	0	0	0	1

- To avoid adding too many dimensions to your dataset. We recommend that you first check the number of values in the column, and bin or quantize the data appropriately.

Next steps

See the set of components available to Azure Machine Learning.

Edit Metadata component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component included in Azure Machine Learning designer.

Use the Edit Metadata component to change metadata that's associated with columns in a dataset. The value and data type of the dataset will change after use of the Edit Metadata component.

Typical metadata changes might include:

- Treating Boolean or numeric columns as categorical values.
- Indicating which column contains the **class** label or contains the values you want to categorize or predict.
- Marking columns as features.
- Changing date/time values to numeric values or vice versa.
- Renaming columns.

Use Edit Metadata anytime you need to modify the definition of a column, typically to meet requirements for a downstream component. For example, some components work only with specific data types or require flags on the columns, such as `IsFeature` or `IsCategorical`.

After you perform the required operation, you can reset the metadata to its original state.

Configure Edit Metadata

1. In Azure Machine Learning designer, add the Edit Metadata component to your pipeline and connect the dataset you want to update. You can find the component in the **Data Transformation** category.
2. Click **Edit column** in the right panel of the component 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.
3. Select the **Data type** option if you need to assign a different data type to the selected columns. You might need to change the data type 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 supported data types are **String**, **Integer**, **Double**, **Boolean**, and **DateTime**.
    - If you select multiple columns, you must apply the metadata changes to *all* selected columns. For example, let's say you choose two or three numeric columns. You can 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 don't specify a new data type, the column metadata is unchanged.
    - The column type and values will change after you perform the Edit Metadata operation. You can recover the original data type at any time by using Edit Metadata to reset the column data type.

> **NOTE**
>
> The **DateTime Format** follows Python built-in datetime format.
> If you change any type of number to the **DateTime** type, leave the **DateTime Format** field blank. Currently it isn't possible to specify the target data format.

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 ensure that the values are used only to group data and not in numeric calculations.

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 in components that operate only on feature columns. By default, all columns are initially treated as features.
   - **Label**: Use this option to mark the label, which is also known as the predictable attribute or target variable. Many components require that exactly one label column is present in the dataset.

     In many cases, Azure Machine Learning can infer that a column contains a class label. By setting this metadata, you can ensure that the column is identified correctly. Setting this option does not change data values. It changes only the way that some machine-learning algorithms handle the data.

   > **TIP**
   >
   > Do you have data that doesn't fit into these categories? For example, your dataset might contain values such as unique identifiers that aren't useful as variables. Sometimes such IDs can cause problems when used in a model.
   >
   > Fortunately, Azure Machine Learning keeps all of your data, so that 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 component. Later you can merge the columns back into the dataset by using the Add Columns component.

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.

     All columns are initially treated as features. For components that perform mathematical operations, you might need to use this option in order 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.

     You currently can't explicitly mark a column as a score in Azure Machine Learning. However, some operations result in a column being flagged as a score internally. Also, a custom R component might output score values.

7. For **New column names**, enter the new name of the selected column or columns.

   - Column names can use only characters that are supported by UTF-8 encoding. Empty strings, nulls, or names that consist entirely of spaces aren't allowed.

- To rename multiple columns, enter the names as a comma-separated list in order of the column indexes.
- All selected columns must be renamed. You can't omit or skip columns.

8. Submit the pipeline.

Next steps

See the set of components available to Azure Machine Learning.

Group Data into Bins component

3/28/2022 • 5 minutes to read • Edit Online

This article describes how to use the Group Data into Bins component in Azure Machine Learning designer, to group numbers or change the distribution of continuous data.

The Group Data into Bins component 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.
- Assign values to bins by using *quantiles*, or percentile ranks.
- 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. It's useful in scenarios like these:

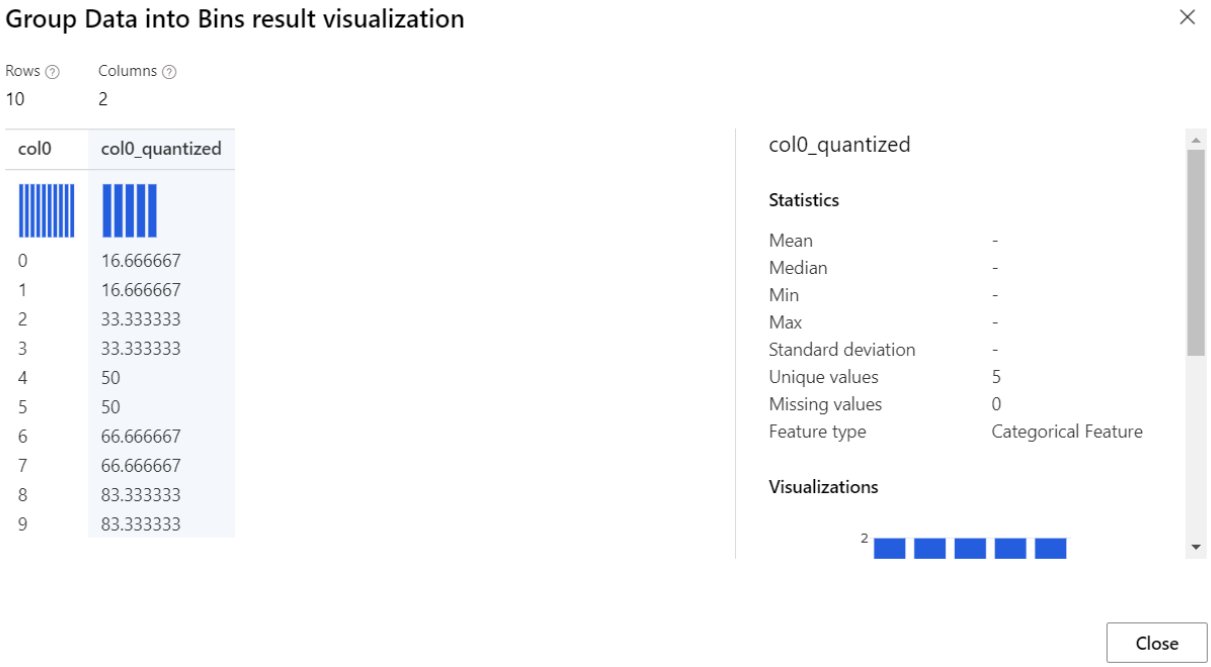
- A column of continuous numbers has too many unique values to model effectively. So you automatically or manually assign the values to groups, to create a smaller set of discrete ranges.
- You want to replace a column of numbers with categorical values that represent specific ranges.

  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 by using the quantiles method.

  With this method, the Group Data into Bins component 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 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.



Because there are so many ways to group data, all customizable, we recommend that you experiment with different methods and values.

How to configure Group Data into Bins

1. Add the **Group Data Into Bins** component to your pipeline in the designer. You can find this component in the category **Data Transformation**.
2. Connect the dataset that has numerical data to bin. Quantization can be applied only to columns that contain numeric data.

   If the dataset contains non-numeric columns, use the Select Columns in Dataset component 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:

   - **Quantiles**: The quantile method assigns values to bins based on percentile ranks. This method 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 ending 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 will have values greater than 0, and one will have values less than or equal to 0. In this case, for bin edges, you enter **0** in **Comma-separated list of bin edges**. The output of the component will be 1 and 2, indicating the bin index for each row value. Note that the comma-separated value list must be in an ascending order, such as 1, 3, 5, 7.

> **NOTE**
>
> *Entropy MDL* mode is defined in Studio (classic) and there's no corresponding open source package which can be leveraged to support in Designer yet.

4. If you're using the **Quantiles** and **Equal Width** binning modes, use the **Number of bins** option to specify how many bins, or *quantiles*, 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. If you need to bin some columns by using a different method, use a separate instance of the Group Data into Bins component for each set of columns.

   > **WARNING**
   >
   > If you choose a column that's not an allowed type, a runtime error is generated. The component 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 before sorting into quantiles. Note that normalizing values transforms the values but doesn't affect the final number of bins.

   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, enter 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. For example, if you enter one bin edge value, two bins will be generated. If you enter two bin edge values, three bins will be generated.

   The values must be sorted in the order that the bins are created, from lowest to highest.

9. Select the **Tag columns as categorical** option to indicate that the quantized columns should be handled as categorical variables.

10. Submit the pipeline.

Results

The Group Data into Bins component returns a dataset in which each element has been binned according to the specified mode.

It also returns a *binning transformation*. That function can be passed to the Apply Transformation component to bin new samples of data by using the same binning mode and parameters.

> **TIP**
>
> If you use binning on your training data, you must use the same binning method on data that you use for testing and prediction. You must also use the same 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. Then apply them to other datasets by using the Apply Transformation component.

Next steps

See the set of components available to Azure Machine Learning.

Join Data

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the **Join Data** component in Azure Machine Learning designer to merge two datasets using a database-style join operation.

How to configure Join Data

To perform a join on two datasets, they should be related by a key column. Composite keys using multiple columns are also supported.

1. Add the datasets you want to combine, and then drag the **Join Data** component into your pipeline.

   You can find the component in the **Data Transformation** category, under **Manipulation**.

2. Connect the datasets to the **Join Data** component.

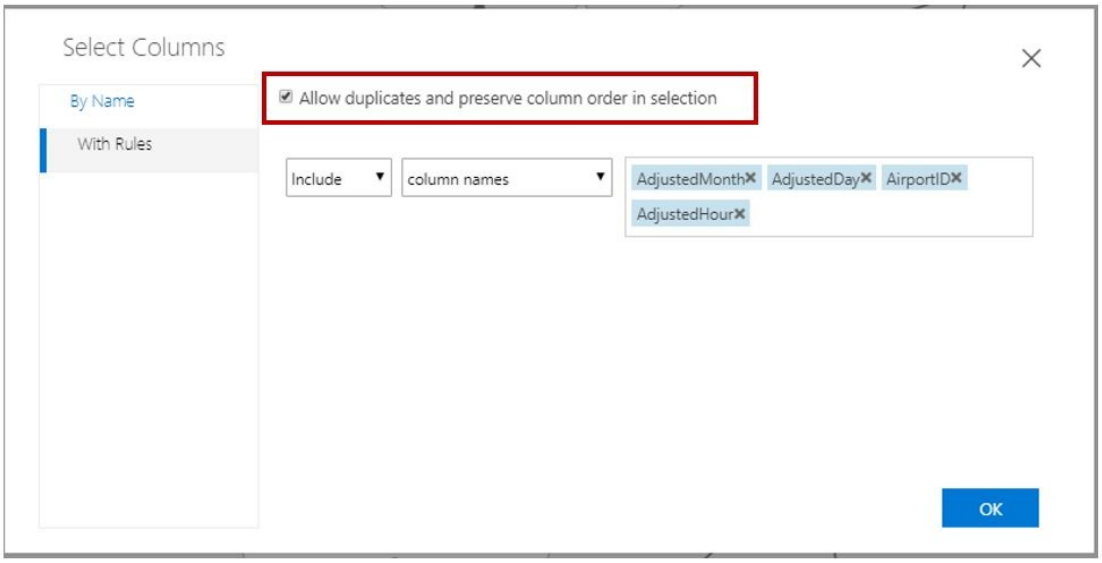
3. Select **Launch column selector** to choose key column(s). Remember to choose columns for both the left and right inputs.

   For a single key:

   Select a single key column for both inputs.

   For a composite key:

   Select all the key columns from left input and right input in the same order. The **Join Data** component will join the tables when all key columns match. Check the option **Allow duplicates and preserve column order in selection** if the column order isn't the same as the original table.



4. Select the **Match case** option if you want to preserve case sensitivity on a text column join.

5. Use the **Join type** dropdown list to specify how the datasets should be combined.

   - **Inner Join**: An *inner join* is the most common 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. You can also 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 either table that have no matching rows in the other, the result includes a row containing missing values.

- **Left Semi-Join**: A *left semi-join* returns only the values from the left table when the values of the key columns match.

6. For the option **Keep right key columns in joined table**:

   - Select this option to view the keys from both input tables.
   - Deselect to only return the key columns from the left input.

7. Submit the pipeline.

8. To view the results, right-click the **Join Data** and select **Visualize**.

Next steps

See the set of components available to Azure Machine Learning.

Normalize Data component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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 component 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 pipeline, 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.

Configure Normalize Data

You can apply only one normalization method at a time using this component. 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** component to your pipeline. You can find the component In Azure Machine Learning, 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 component 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 function 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:

     $$z = \frac{\mathrm{x} - mean(x)}{stdev(x)}$$

     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:

     $$z = \frac{\mathrm{x} - min(x)}{[\max(x) - min(x)]}$$

   - **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:

     $$z = Lognormal.CDF(x;;\mu,\sigma)$$

     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. Submit the pipeline, or double-click the **Normalize Data** component and select **Run Selected**.

Results

The **Normalize Data** component generates two outputs:

- To view the transformed values, right-click the component, and select **Visualize**.

  By default, values are transformed in place. If you want to compare the transformed values to the original values, use the Add Columns component 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 dataset, select the component, and select **Register dataset** under the **Outputs** tab in the right panel.

  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.

Next steps

See the set of components available to Azure Machine Learning.

Partition and Sample component

3/28/2022 • 7 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use the Partition and Sample component 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 component 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 you randomly assign 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 pipeline, 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.

Configure the component

This component supports the following methods for dividing your data into partitions or for sampling. Choose the method first, and then set additional options that the method requires.

- Head
- Sampling
- Assign to folds
- Pick fold

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 a pipeline on a small number of rows, and you don't need the data to be balanced or sampled in any way.

1. Add the **Partition and Sample** component to your pipeline in the interface, and connect the dataset.
2. **Partition or sample mode**: Set this option to **Head**.
3. **Number of rows to select**: Enter the number of rows to return.

   The number of rows 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. Submit the pipeline.

The component outputs a single dataset that contains 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. It's useful if you want to create a smaller representative sample dataset for testing.

1. Add the **Partition and Sample** component to your pipeline, and connect the dataset.
2. **Partition or sample mode**: Set this option to **Sampling**.
3. **Rate of sampling**: Enter 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, enter `0.5` to indicate that the sampling rate should be 50 percent.

   The rows of the input dataset are shuffled and selectively placed in the output dataset, according to the specified ratio.
4. **Random seed for sampling**: Optionally, enter 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 value can lead to slightly different results each time you run the pipeline.
5. **Stratified split for sampling**: Select this option if it's important that the rows in the dataset are 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:

   a. All input rows are grouped (stratified) by the values in the specified strata column.

   b. Rows are shuffled within each group.

   c. Each group is selectively added to the output dataset to meet the specified ratio.
6. Submit the pipeline.

   With this option, the component outputs a single dataset that contains a representative sampling of the data. The remaining, unsampled portion of the dataset is not output.

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** component to your pipeline, 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 don't 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 don't select this option, rows are assigned to folds through the round-robin method.

5. **Random seed**: Optionally, enter 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, by 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, enter a whole number in the **Specify number of folds to split evenly into** box.
   - **Partition with customized proportions**: Use this option to specify the size of each partition as a comma-separated list.

     For example, assume that you want to create three partitions. The first partition will contain 50 percent of the data. The remaining two partitions will each contain 25 percent of the data. In the **List of proportions separated by comma** box, enter 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 enter the values **.2** and **.3**, a third partition is created to hold 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 pipeline.
7. **Stratified split**: Select this option if you want the rows to be stratified when split, and then choose the *strata column*.
8. Submit the pipeline.

   With this option, the component outputs multiple datasets. The datasets are partitioned according to the rules that you specified.

Use data from a predefined partition

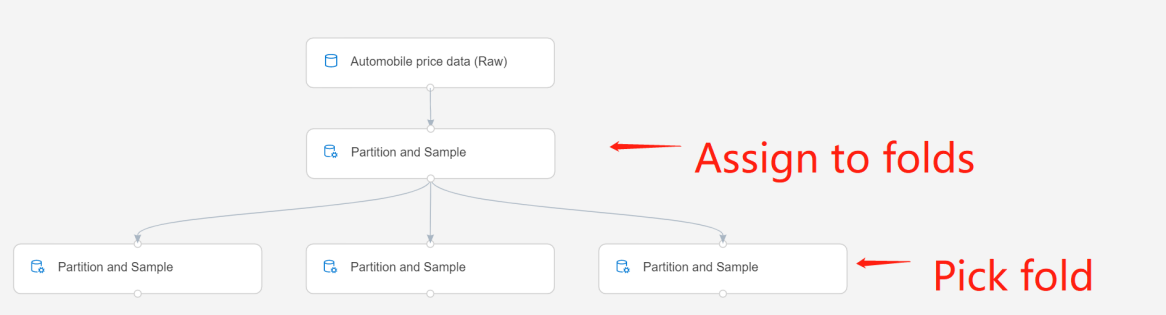
Use this option when you have divided a dataset into multiple partitions and now want to load each partition in turn for further analysis or processing.

1. Add the **Partition and Sample** component to the pipeline.
2. Connect the component 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 entering 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 enter 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, enter 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're working with multiple partitions, you must add more instances of the **Partition and Sample** component to handle each partition.

For example, the **Partition and Sample** component in the second row is set to **Assign to Folds**, and the component in the third row is set to **Pick Fold**.



6. Submit the pipeline.

   With this option, the component outputs a single dataset that contains only the rows assigned to that fold.

> **NOTE**
> You can't view the fold designations directly. They're present only in the metadata.

Next steps

See the set of components available to Azure Machine Learning.

Remove Duplicate Rows component

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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	
6	F.A.M.	M	NaN	
7	F.A.M.	M	NaN	

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 files 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. If any row has missing value for **keys**, they will not be considered duplicate rows. For example, if Gender and Age are set as Keys in above table, row 6 and 7 are not duplicate rows given they have missing value in Age.

When you run the component, 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 component 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 component to your pipeline. You can find the **Remove Duplicate Rows** component 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 unselected 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 equalities 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.
5. Submit the pipeline.
6. To review the results, right-click the component, and select **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 component.

Next steps

See the set of components available to Azure Machine Learning.

SMOTE

3/28/2022 • 5 minutes to read • Edit Online

This article describes how to use the SMOTE component in Azure Machine Learning designer to increase the number of underrepresented cases in a dataset that's 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 component to a dataset that's *imbalanced*. There are many reasons why a dataset might be imbalanced. For example, the category you're targeting might be rare in the population, or the data might be difficult to collect. Typically, you use SMOTE when the *class* that you want to analyze is underrepresented.

The component returns a dataset that contains the original samples. It also returns a number of synthetic minority samples, depending on the percentage that you specify.

More about SMOTE

Synthetic Minority Oversampling Technique (SMOTE) is a statistical technique for increasing the number of cases in your dataset in a balanced way. The component 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. The algorithm then 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 percent of the cases have the target value A (the minority class), and 99 percent 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 component'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 designer.

If you add the dataset to a pipeline and select **Visualize** on the dataset's output, you can see that of the 748 rows or cases in the dataset, 570 cases (76 percent) are of Class 0, and 178 cases (24 percent) are of Class 1. Although this result isn't terribly imbalanced, Class 1 represents the people who donated blood, so 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**, by using multiples of 100, as follows:

	CLASS 0	CLASS 1	TOTAL
Original dataset (equivalent to **SMOTE percentage** = **0**)	570 76%	178 24%	748

	CLASS 0	CLASS 1	TOTAL
SMOTE percentage = 100	570 62%	356 38%	926
SMOTE percentage = 200	570 52%	534 48%	1,104
SMOTE percentage = 300	570 44%	712 56%	1,282

WARNING

Increasing the number of cases by using SMOTE is not guaranteed to produce more accurate models. Try pipelining 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 component to your pipeline. You can find the component under **Data Transformation components**, in the **Manipulation** category.

2. Connect the dataset that 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 component. You can then isolate the columns that you want to use before using SMOTE.

   Otherwise, creation of new cases through SMOTE is based on *all* the columns that you provide as inputs. At least one column of the feature columns is numeric.

3. Ensure that the column that contains the label, or target class, is selected. SMOTE accepts only binary labels.

4. The SMOTE component automatically identifies the minority class in the label column, and then gets all examples for the minority class. All columns can't have NaN values.

5. In the **SMOTE percentage** option, enter a whole number that indicates the target percentage of minority cases in the output dataset. For example:

   - You enter **0**. The SMOTE component returns exactly the same dataset that you provided as input. It adds no new minority cases. In this dataset, the class proportion has not changed.
   - You enter **100**. The SMOTE component generates new minority cases. It adds 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 changed.
   - You enter **200**. The component 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. The number of minority cases is increased until 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 in building new cases. A nearest neighbor is a row of data (a case) that's similar to a 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. Enter a value in the **Random seed** box if you want to ensure the same results over runs of the same pipeline, with the same data. Otherwise, the component generates a random seed based on processor clock values when the pipeline is deployed. The generation of a random seed can cause slightly different results over runs.

8. Submit the pipeline.

   The output of the component is a dataset that contains the original rows plus a number of added rows with minority cases.

Technical notes

- When you're publishing a model that uses the **SMOTE** component, remove **SMOTE** from the predictive pipeline before it's published as a web service. The reason is that SMOTE is intended for improving a model during training, not for scoring. You might get an error if a published predictive pipeline contains the SMOTE component.
- You can often get better results if you clean missing values or apply other transformations to fix data before you apply SMOTE.
- Some researchers have investigated whether SMOTE is effective on high-dimensional or sparse data, such as data 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:
  - Methods for oversampling the minority cases or undersampling the majority cases.
  - Ensemble techniques that help the learner directly by using clustering, bagging, or adaptive boosting.

Next steps

See the set of components available to Azure Machine Learning.

Select Columns Transform

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the Select Columns Transform component in Azure Machine Learning designer. The purpose of the Select Columns Transform component is to ensure that a predictable, consistent set of columns is used in downstream machine learning operations.

This component is helpful for tasks such as scoring, which require specific columns. Changes in the available columns might break the pipeline or change the results.

You use Select Columns Transform to create and save a set of columns. Then, use the Apply Transformation component to apply those selections to new data.

How to use Select Columns Transform

This scenario assumes that you want 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 component to capture the column selections and apply them elsewhere in the pipeline.

1. Add an input dataset to your pipeline in the designer.
2. Add an instance of Filter Based Feature Selection.
3. Connect the components and configure the feature selection component to automatically find a 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 based on the values in the column, you can't know in advance which columns might be available for input to Train Model.

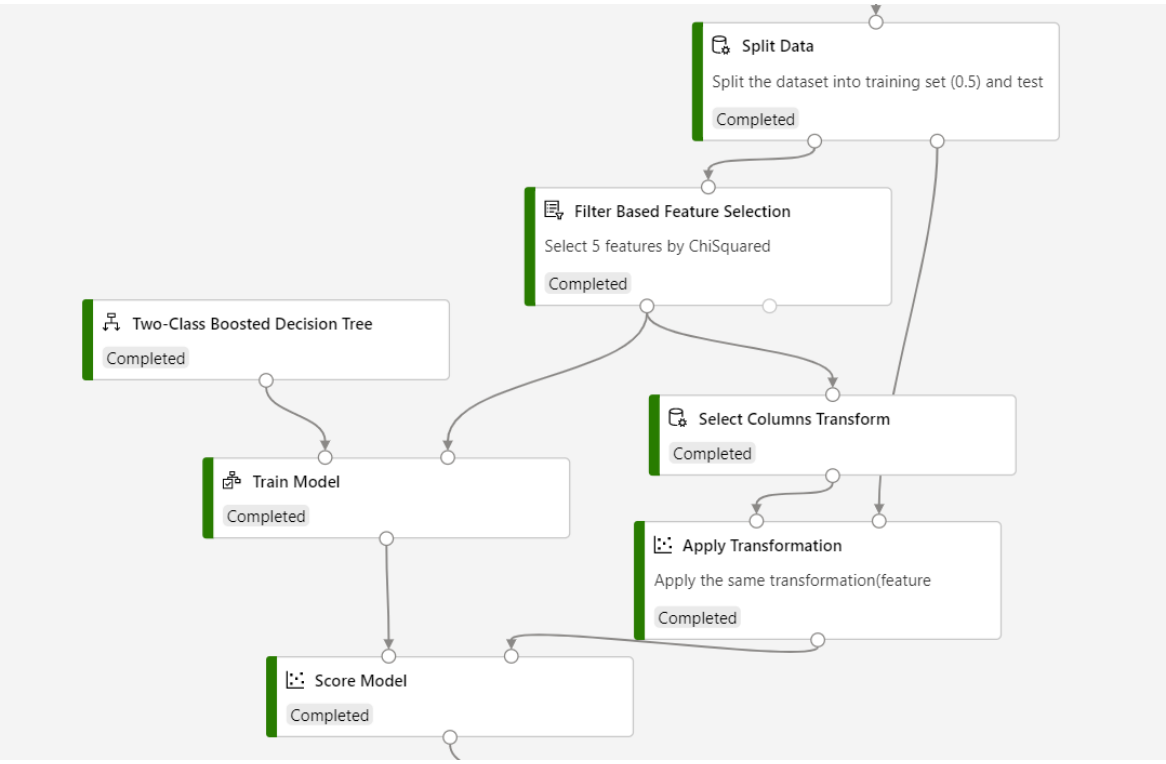
5. Attach an instance of the Select Columns Transform component.

   This step generates a column selection as a transformation that can be saved or applied to other datasets. This step ensures that the columns identified in feature selection are saved for other components to reuse.

6. Add the Score Model component.

   Do not connect the input dataset. Instead, add the Apply Transformation component, and connect the output of the feature selection transformation.

   The pipeline structure should be like following:



IMPORTANT

You can't expect to apply Filter Based Feature Selection to the scoring dataset and get the same results. Because feature selection is based on values, it might choose a different set of columns, which would cause the scoring operation to fail.

7. Submit the pipeline.

This process of saving and then applying a column selection ensures that the same data schema is available for training and scoring.

Next steps

See the set of components available to Azure Machine Learning.

Select Columns in Dataset component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to choose a subset of columns to use in downstream operations. The component 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 component is 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

This component 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 component 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 components 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 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

component 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.

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 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`.

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`

Next steps

See the set of components available to Azure Machine Learning.

Split Data component

3/28/2022 • 5 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use the Split Data component to divide a dataset into two distinct sets.

This component is useful when you need to separate data into training and testing sets. You can also customize the way that data is divided. Some options support randomization of data. Others are tailored for a certain data type or model type.

Configure the component

> **TIP**
> Before you choose the splitting mode, read all options to determine the type of split you need. If you change the splitting mode, all other options might be reset.

1. Add the **Split Data** component to your pipeline in the designer. You can find this component 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.
   - **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. 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.
   - **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're analyzing sentiment, you can check for the presence of a particular product name in a text field. You can then divide the dataset into rows with the target product name and rows without the target product name.
   - **Relative Expression Split**: Use this option whenever you want to apply a condition to a number column. The number can be a date/time field, a column that contains age or dollar amounts, or even a percentage. For example, you might want to divide your dataset based on the cost of the items, group people by age ranges, or separate data by a calendar date.

Split rows

1. Add the Split Data component to your pipeline in the designer, and connect the dataset that you want to split.
2. For **Splitting mode**, select **Split Rows**.
3. **Fraction of rows in the first output dataset**: Use this option to determine how many rows will go into the first (left side) output. All other rows will go into the second (right side) output.

   The ratio represents the percentage of rows sent to the first output dataset, so you must enter a decimal

number between 0 and 1.

For example, if you enter **0.75** as the value, the dataset will be split 75/25. In this split, 75 percent of the rows will be sent to the first output dataset. The remaining 25 percent will be 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 you're creating training and test datasets.

5. **Random Seed**: This parameter will be ignored if **Randomized split** is set to false. Otherwise enter a non-negative integer value to start the pseudorandom sequence of instances to be used. This default seed is used in all components that generate random numbers.

   Specifying a seed makes the results reproducible. If you need to repeat the results of a split operation, you should specify the same seed number for the random number generator.

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 to some other column (such as gender).

7. Submit the pipeline.

Select a regular expression

1. Add the Split Data component to your pipeline, and connect it as input to the dataset that you want to split.

2. For **Splitting mode**, select **Regular expression split**.

3. In the **Regular expression** box, enter a valid regular expression.

   The regular expression should follow Python syntax for regular expressions.

4. Submit the pipeline.

   Based on the regular expression that you provide, the dataset is divided into two sets of rows: rows with values that match the expression and all remaining rows.

The following examples demonstrate how to divide a dataset by 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`. It 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. The position is 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.

Select a relative expression

1. Add the Split Data component to your pipeline, and connect it as input to the dataset that you want to split.
2. For **Splitting mode**, select **Relative Expression**.
3. In the **Relational expression** box, enter an expression that performs a comparison operation on a single column.

   For **Numeric column**:

   - The column contains numbers of any numeric data type, including date and time data types.
   - The expression can reference a maximum of one column name.
   - Use the ampersand character, `&`, for the AND operation. Use the pipe character, `|`, for the OR operation.
   - The following operators are supported: `<`, `>`, `<=`, `>=`, `==`, `!=`.
   - You can't group operations by using `(` and `)`.

   For **String column**:

   - The following operators are supported: `==`, `!=`.
4. Submit the pipeline.

   The expression divides the dataset into two sets of rows: rows with values that meet the condition, and all remaining rows.

The following examples demonstrate how to divide a dataset by using the **Relative Expression** option in the **Split Data** component.

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. The format of dates in the data column must be consistent.

For example, in a date column that uses the format `mmddyyyy`, the expression should be something like this:

```
\"Date" > 1/1/2010
```

Column index

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
```

Next steps

See the set of components available to Azure Machine Learning.

Filter Based Feature Selection

3/28/2022 • 6 minutes to read • Edit Online

This article describes how to use the Filter Based Feature Selection component in Azure Machine Learning designer. This component helps you 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. The goal is to determine which columns are more predictive of the output. The Filter Based Feature Selection component provides multiple feature selection algorithms to choose from. The component includes correlation methods such as Pearson correlation and chi-squared values.

When you use the Filter Based Feature Selection component, you provide a dataset and identify the column that contains the label or dependent variable. You then specify a single method to use in measuring feature importance.

The component 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 filter-based feature selection is

This component for feature selection is called "filter-based" because you use the selected metric to find irrelevant attributes. You then filter out redundant columns from your model. You choose a single statistical measure that suits your data, and the component 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 component 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's applied. You can find additional requirements for using each metric in the technical notes and in the instructions for configuring each component.

- **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. Changes of scale in the two variables don't affect the coefficient.

- **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.

> **TIP**
> If you need a different option for the custom feature selection method, use the Execute R Script component.

How to configure Filter-Based Feature Selection

You choose a standard statistical metric. The component computes the correlation between a pair of columns: the label column and a feature column.

1. Add the Filter-Based Feature Selection component to your pipeline. You can find it in the **Feature Selection** category in the designer.
2. Connect an input dataset that contains at least two columns that are potential features.

   To ensure that a column is analyzed and a feature score is generated, use the Edit Metadata component to set the **IsFeature** attribute.

   > **IMPORTANT**
   > Ensure that the columns that you're providing as input are potential features. For example, a column that contains a single value has no information value.
   >
   > If you know that some columns would make bad features, you can remove them from the column selection. You can also use the Edit Metadata component 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.
Chi squared	Labels and features can be text or numeric. Use this method for computing feature importance for two categorical columns.

   > **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 columns that were previously marked as features.

   If you clear this option, the component 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**, select **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 component returns a design-time error if you choose no label column or multiple label columns.

6. For **Number of desired features**, enter the number of feature columns that you want returned as a result:

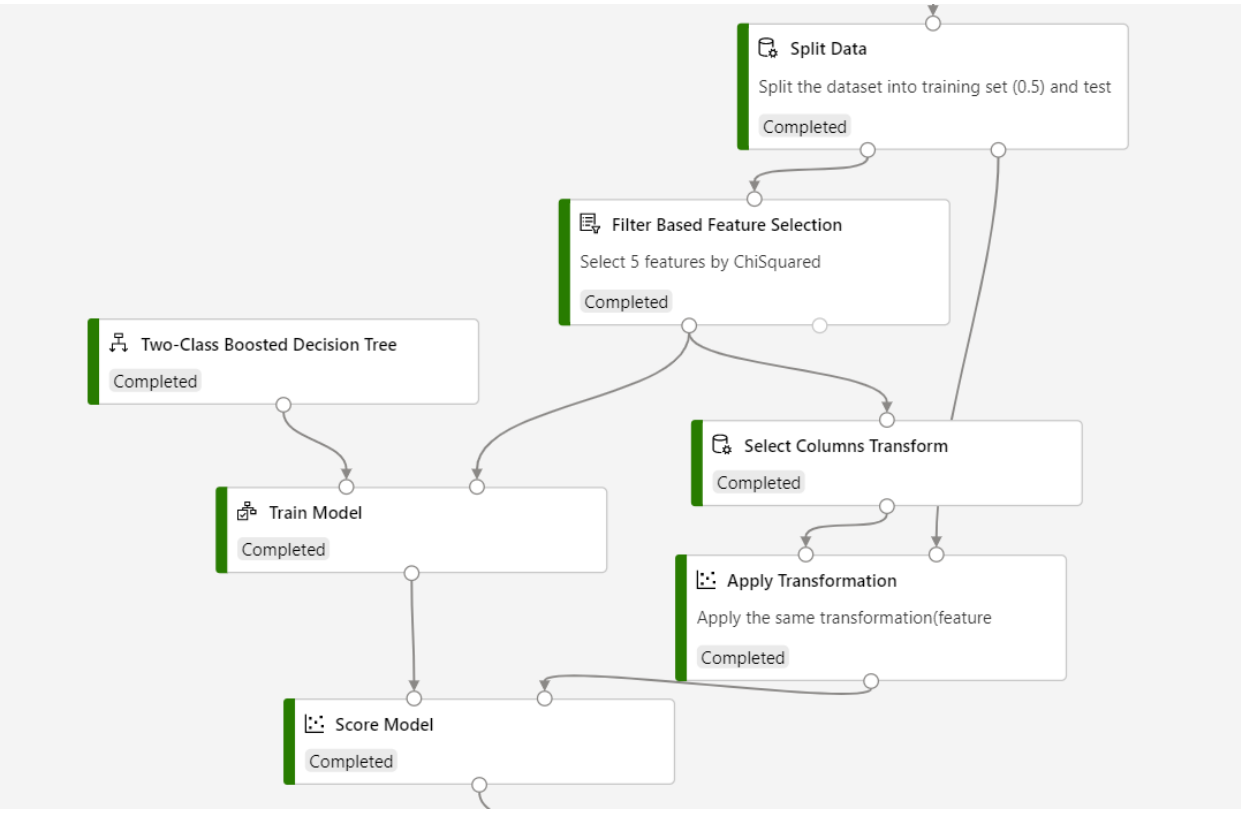
   - The minimum number of features that you can specify is one, but we recommend that you

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 features with zero scores are returned.
- If you specify fewer result columns than there are feature columns, the features are ranked by descending score. Only the top features are returned.

7. Submit the pipeline.

> **IMPORTANT**
>
> If you are going to use **Filter Based Feature Selection** in inference, you need to use Select Columns Transform to store the feature selected result and Apply Transformation to apply the feature selected transformation to the scoring dataset.
>
> Refer to the following screenshot to build your pipeline, to ensure that column selections are the same for the scoring process.



Results

After processing is complete:

- To see a complete list of the analyzed feature columns and their scores, right-click the component and select **Visualize**.
- To view the dataset based on your feature selection criteria, right-click the component and select **Visualize**.

If the dataset contains fewer columns than you expected, check the component settings. Also check 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.

Technical notes

Implementation details

If you use Pearson 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 can't be generated for any column that's designated as a **Label** or **Score** column.
- If you try to use a scoring method with a column of a data type that the method doesn't support, the component will raise an error. Or, a zero score will be assigned to the column.
- If a column contains logical (true/false) values, they're processed as `True = 1` and `False = 0`.
- A column can't be a feature if it has been designated as a **Label** or a **Score**.

How missing values are handled

- You can't specify as a target (label) column any column that has all missing values.
- If a column contains missing values, the component ignores them when it's computing the score for the column.
- If a column designated as a feature column has all missing values, the component assigns a zero score.

Next steps

See the set of components available to Azure Machine Learning.

Permutation Feature Importance

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the Permutation Feature Importance component in Azure Machine Learning designer, 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 component, feature values are randomly shuffled, one column at a time. The performance of the model is measured before and after. You can choose one of the standard metrics to measure performance.

The scores that the component returns represent the *change* in the performance of a trained model, after permutation. Important features are usually more sensitive to the shuffling process, so they'll result in higher importance scores.

This article provides an overview of the permutation feature, its theoretical basis, and its applications in machine learning: Permutation Feature Importance.

How to use Permutation Feature Importance

Generating 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 component to your pipeline. You can find this component in the **Feature Selection** category.
2. Connect a trained model to the left input. The model must be a regression model or a classification model.
3. On the right input, connect a dataset. Preferably, choose one that's different from the dataset that you used for training the model. This dataset is used for scoring based on the trained model. It's also used for evaluating the model after feature values have changed.
4. For **Random seed**, enter a value to use as a 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 pipeline.
5. For **Metric for measuring performance**, select a single metric to use when you're computing model quality after permutation.

   Azure Machine Learning designer supports the following metrics, depending on whether you're evaluating a classification or regression model:

   - **Classification**

     Accuracy, Precision, Recall

   - **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're calculated, see Evaluate Model.
6. Submit the pipeline.

7. The component outputs a list of feature columns and the scores associated with them. The list is ranked in descending order of the scores.

Technical notes

Permutation Feature Importance works by randomly changing the values of each feature column, one column at a time. It then evaluates the model.

The rankings that the component provides are often different from the ones you get from Filter Based Feature Selection. Filter Based Feature Selection calculates scores *before* a model is created.

The reason for the difference is that Permutation Feature Importance doesn't measure the association between a feature and a target value. Instead, it captures how much influence each feature has on predictions from the model.

Next steps

See the set of components available to Azure Machine Learning.

Summarize Data

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component of Azure Machine Learning designer.

Use the Summarize Data component to create a set of standard statistical measures that describe each column in the input table.

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 component calculates the important scores for each column, and returns a row of summary statistics for each variable (data column) provided as input.

How to configure Summarize Data

1. Add the **Summarize Data** component to your pipeline. You can find this component in the **Statistical Functions** category in the designer.
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 component to project a subset of columns to work with.
3. No additional parameters are required. By default, the component 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. Submit the pipeline.

Results

The report from the component can include the following 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

COLUMN NAME	DESCRIPTION
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
P99.5	99.5% percentile

Technical notes

- 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 component 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.

Next steps

See the set of components available to Azure Machine Learning.

Boosted Decision Tree Regression component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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 component is based LightGBM algorithm.

This regression method is a supervised learning method, and therefore requires a *labeled dataset*. The label column must contain numerical values.

> **NOTE**
> Use this component only with datasets that use numerical variables.

After you have defined the model, train it by using the Train Model.

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, 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** component to your pipeline. You can find this component 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.

   If you set the value to 1; however, 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. Train the model:

   - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
   - 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 default value in the single parameter list.
   >
   > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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. Submit the pipeline.

Results

After training is complete:

- To use the model for scoring, connect Train Model to Score Model, to predict values for new input examples.
- To save a snapshot of the trained model, select **Outputs** tab in the right panel of **Trained model** and click **Register dataset** icon. The copy of the trained model will be saved as a component in the component tree and will not be updated on successive runs of the pipeline.

Next steps

See the set of components available to Azure Machine Learning.

Decision Forest Regression component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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 component. The trained model can then be used to make predictions.

How it works

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** component to the pipeline. You can find the component in the designer under **Machine Learning**, **Initialize Model**, and **Regression**.
2. Open the component 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 Gaussian distributions given by combining all distributions 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**: 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.

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**
   > If you 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 five cases that meet the same conditions.

8. Train the model:

   - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
   - 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 default value in the single parameter list.
   >
   > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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. Submit the pipeline.

Results

After training is complete:

- To save a snapshot of the trained model, select the training component, then switch to **Outputs** tab in the right panel. Click on the icon **Register model**. You can find the saved model as a component in the

component tree.

Next steps

See the set of components available to Azure Machine Learning.

Fast Forest Quantile Regression

3/28/2022 • 4 minutes to read • Edit Online

This article describes a module in Azure Machine Learning designer.

Use this component to create a fast forest quantile regression model in a pipeline. Fast forest 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. Simply put, 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 component, have the additional advantage that they can be used to predict non-parametric distributions.

How to configure Fast Forest Quantile Regression

1. Add the **Fast Forest Quantile Regression** component to your pipeline in the designer. You can find this component under **Machine Learning Algorithms**, in the **Regression** category.
2. In the right pane of the **Fast Forest Quantile Regression** component, 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. When you train the model, use Train Model.
   - **Parameter Range**: If you are not sure of the best parameters, do a parameter sweep using the Tune Model Hyperparameters component. The trainer iterates over multiple values you specify to find the optimal configuration.
3. **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. **Number of Leaves**, type the maximum number of leaves, or terminal nodes, that can be created in any tree.

5. **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. **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. **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.

8. **Quantiles to be estimated**, type a semicolon-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`.

9. 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.

10. Connect the training dataset and the untrained model to one of the training components:

    - If you set **Create trainer mode** to **Single Parameter**, use the Train Model component.
    - If you set **Create trainer mode** to **Parameter Range**, use the Tune Model Hyperparameters component.

    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 component, 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. Submit the pipeline.

Results

After training is complete:

- To save a snapshot of the trained model, select the training component, then switch to **Outputs+logs** tab in the right panel. Click on the icon **Register dataset**. You can find the saved model as a component in the component tree.

Next steps

See the set of components available to Azure Machine Learning.

Linear Regression component

3/28/2022 • 6 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to create a linear regression model for use in a pipeline. Linear regression attempts to establish a linear relationship between one or more independent variables and a numeric outcome, or dependent variable.

You use this component 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.

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. Simply put, regression refers to prediction of a numeric target. Linear regression is still a good choice when you want a 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 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.

- The classic regression problem involves a single independent variable and a dependent variable. This is called *simple regression*. This component 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** component can solve these problems, as can most of the other regression components.
- *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 component 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.

Configure Linear Regression

This component supports two methods for fitting a regression model, with different options:

- Fit a regression model using ordinary least squares

  For small datasets, it is best to select ordinary least squares. This should give similar results to Excel.

- 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.

Create a regression model using ordinary least squares

1. Add the **Linear Regression Model** component to your pipeline in the designer.

   You can find this component in the **Machine Learning** category. Expand **Initialize Model**, expand **Regression**, and then drag the **Linear Regression Model** component to your pipeline.

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 pipeline. Otherwise, the default is to use a value from the system clock.

6. Add the Train Model component to your pipeline, and connect a labeled dataset.

7. Submit the pipeline.

Results for ordinary least squares model

After training is complete:

- To make predictions, connect the trained model to the Score Model component, along with a dataset of new values.

Create a regression model using online gradient descent

1. Add the **Linear Regression Model** component to your pipeline in the designer.

   You can find this component in the **Machine Learning** category. Expand **Initialize Model**, expand **Regression**, and drag the **Linear Regression Model** component to your pipeline

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**: 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.

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 component 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, **Decrease learning rate**, if you want the learning rate to decrease as iterations progress.
9. 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 pipeline.
10. Train the model:

    - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
    - 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 default value in the single parameter list.
    >
    > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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. Submit the pipeline.

Results for online gradient descent

After training is complete:

- To make predictions, connect the trained model to the Score Model component, together with new input data.

Next steps

See the set of components available to Azure Machine Learning.

Neural Network Regression component

3/28/2022 • 4 minutes to read • Edit Online

Creates a regression model using a neural network algorithm

Category: Machine Learning / Initialize Model / Regression

Component overview

This article describes a component in Azure Machine Learning designer.

Use this component 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. The trained model can then be used to predict values for the new input examples.

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 component 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** component to your pipeline in the designer. You can find this component 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**: 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. 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 momentum**, type a value to apply during learning as a weight on nodes from previous iterations.

8. 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 pipeline.

9. 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 pipeline.

10. Connect a training dataset and train the model:

    - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
    - 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 default value in the single parameter list.
    >
    > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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. Submit the pipeline.

Results

After training is complete:

- To save a snapshot of the trained model, select the **Outputs** tab in the right panel of the **Train model**

component. Select the **Register dataset** icon to save the model as a reusable component.

Next steps

See the set of components available to Azure Machine Learning.

Poisson Regression

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to create a Poisson regression model in a pipeline. Poisson regression is intended for predicting numeric values, typically counts. Therefore, you should use this component 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 other regression components in the designer.

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** component to your pipeline in designer. You can find this component under **Machine Learning Algorithms**, 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 right pane of the **Poisson Regression** component, 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 component. 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 component, 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 components:

   - If you set **Create trainer mode** to **Single Parameter**, use the Train Model component.
   - If you set **Create trainer mode** to **Parameter Range**, use the Tune Model Hyperparameters component.

   > **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 component, 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. Submit the pipeline.

Results

After training is complete:

- To save a snapshot of the trained model, select the training component, then switch to **Outputs+logs** tab in the right panel. Click on the icon **Register dataset**. You can find the saved model as a component in the component tree.

Next steps

See the set of components available to Azure Machine Learning.

Component: K-Means Clustering

3/28/2022 • 6 minutes to read • Edit Online

This article describes how to use the *K-Means Clustering* component in Azure Machine Learning designer to create an untrained K-means clustering model.

K-means is one of the simplest and the best known *unsupervised* learning algorithms. You can use the algorithm for a variety of machine learning tasks, such as:

- Detecting abnormal data.
- Clustering text documents.
- Analyzing datasets before you use other classification or regression methods.

To create a clustering model, you:

- Add this component to your pipeline.
- Connect a dataset.
- Set parameters, such as the number of clusters you expect, the distance metric to use in creating the clusters, and so forth.

After you've configured the component hyperparameters, you connect the untrained model to the Train Clustering Model. 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.

Understand K-means clustering

In general, clustering uses iterative techniques to group cases in a dataset into clusters that possess 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 by using the K-means method, you must specify a target number *k* that indicates the number of *centroids* you want in the model. The centroid is a point that's 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 it processes the training data, the K-means algorithm begins with an initial set of randomly chosen centroids. Centroids serve as starting points for the clusters, and they apply Lloyd's algorithm to iteratively refine their locations. The K-means algorithm stops building and refining clusters when it meets one or more of these conditions:

- The centroids stabilize, meaning that the 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 you've completed the training phase, you use the Assign Data to Clusters component to assign new cases to one of the clusters that you found by using the K-means algorithm. You perform cluster assignment 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.

Configure the K-Means Clustering component

1. Add the **K-Means Clustering** component to your pipeline.
2. To specify how you want the model to be trained, select 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.
3. For **Number of centroids**, type the number of clusters you want the algorithm to begin with.

   The model isn't guaranteed to produce exactly this number of clusters. The algorithm starts with this number of data points and iterates to find the optimal configuration. You can refer to the source code of sklearn.
4. The properties **Initialization** is used to specify the algorithm that's used to define the initial cluster configuration.
   - **First N**: Some initial number of data points are chosen from the dataset and used as the initial means.

     This method is 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.

     This method is 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.
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.
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.
7. For **Iterations**, type the number of times the algorithm should iterate over the training data before it finalizes the selection of centroids.

   You can adjust this parameter to balance accuracy against training time.
8. For **Assign label mode**, choose an option that specifies how a label column, if it's 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 the 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. Select the **Normalize features** option if you want to normalize features before training.

   If you apply normalization, before training, the data points are normalized to `[0,1]` by MinMaxNormalizer.

10. 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 component.

Results

After you've 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 component:

1. Select the **Train Clustering Model** component and open the right panel.
2. Select **Outputs** tab. Select the **Register dataset** icon to save a copy of the trained model.

The saved model represents the training data at the time you saved the model. If you later update the training data used in the pipeline, it doesn't update the saved model.

See the clustering result dataset

If you used the Train Clustering Model component:

1. Right-click the **Train Clustering Model** component.
2. Select **Visualize**.

Tips for generating the best clustering model

It is known that the *seeding* process that's 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, it might have only one point.

You can avoid this problem in a couple of ways:

- Change the number of centroids and try multiple seed values.
- Create multiple models, varying the metric or iterating more.

In general, with clustering models, it's possible that any given configuration will result in a locally optimized set of clusters. In other words, the set of clusters that's returned by the model suits only the current data points and isn't generalizable to other data. If you use a different initial configuration, the K-means method might find a different, superior, configuration.

Next steps

See the set of components available to Azure Machine Learning.

Multiclass Boosted Decision Tree

3/28/2022 • 3 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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 ensemble of trees together.

How to configure

This component creates an untrained classification model. Because classification is a supervised learning method, you need a *labeled dataset* that includes a label column with a value for all rows.

You can train this type of model by using the Train Model.

1. Add the **Multiclass Boosted Decision Tree** component to your pipeline.
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**: 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** limits 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 achieve higher precision, at the risk of overfitting and longer training time.
4. **Minimum number of samples per leaf node** indicates 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 five cases that meet the same conditions.
5. **Learning rate** defines the step size while learning. Enter a number between 0 and 1.

   The learning rate determines how fast or slow the learner converges on an optimal solution. If the step size is too large, 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** indicates 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.
7. **Random number seed** optionally sets 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 42. Successive runs using different random seeds can have different

results.

8. Train the model:

   - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
   - 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 default value in the single parameter list.
>
> If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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.

Next steps

See the set of components available to Azure Machine Learning.

Multiclass Decision Forest component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to create a machine learning model based on the *decision forest* algorithm. A decision forest is an ensemble model that 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 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

1. Add the **Multiclass Decision Forest** component to your pipeline in the designer. You can find this component under **Machine Learning**, **Initialize Model**, and **Classification**.
2. Double-click the component 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.
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**: 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.

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.

   If you 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 five cases that meet the same conditions.

9. Connect a labeled dataset, and train the model:

   - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
   - 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 default value in the single parameter list.
   >
   > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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.

10. Submit the pipeline.

Next steps

See the set of components available to Azure Machine Learning.

Multiclass Logistic Regression component

3/28/2022 • 3 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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 component such as Train Model. The trained model can then be used to predict values for new input examples.

Azure Machine Learning also provides a Two-Class Logistic Regression component, which is suited for classification of binary or dichotomous variables.

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 popular for classification tasks. The algorithm predicts the probability of occurrence of an event by fitting data to a logistic function.

In multiclass logistic regression, the classifier can be used to predict multiple outcomes.

Configure a multiclass logistic regression

1. Add the **Multiclass Logistic Regression** component to the pipeline.
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**: 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. **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. **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 pipeline.

6. Connect a labeled dataset, and train the model:

   - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
   - 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 default value in the single parameter list.
   >
   > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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.

7. Submit the pipeline.

Next steps

See the set of components available to Azure Machine Learning.

Multiclass Neural Network component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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. The trained model can then be used to predict values for the new input examples.

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 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.

Configure Multiclass Neural Network

1. Add the **MultiClass Neural Network** component to your pipeline in the designer. You can find this component 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**: 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. **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.
4. **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.
5. **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.
6. **Number of learning iterations**: Specify the maximum number of times the algorithm should process the training cases.
7. **The initial learning weights diameter**: Specify the node weights at the start of the learning process.
8. **The momentum**: Specify a weight to apply during learning to nodes from previous iterations.
9. **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 pipeline.
10. **Random number seed**: Type a value to use as the seed, if you want to ensure repeatability across runs of the same pipeline.
11. Train the model:

    - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
    - 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 default value in the single parameter list.
    >
    > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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 save a snapshot of the trained model, select the **Outputs** tab in the right panel of the **Train model** component. Select the **Register dataset** icon to save the model as a reusable component.

Next steps

See the set of components available to Azure Machine Learning.

One-vs-All Multiclass

3/28/2022 • 3 minutes to read • Edit Online

This article describes how to use the One-vs-All Multiclass component in Azure Machine Learning designer. The goal is to create a classification model that can predict multiple classes, by using the *one-versus-all* approach.

This component 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-versus-all models

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). But even binary classification algorithms can be adapted for multi-class classification tasks through a variety of strategies.

This component implements the one-versus-all method, in which a binary model is created for each of the multiple output classes. The component assesses each of these binary models for the individual classes against its complement (all other classes in the model) as though it's a binary classification issue. In addition to its computational efficiency (only `n_classes` classifiers are needed), one advantage of this approach is its interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier. This is the most commonly used strategy for multiclass classification and is a fair default choice. The component then performs prediction by running these binary classifiers and choosing the prediction with the highest confidence score.

In essence, the component creates an ensemble of individual models and then merges the results, to create a single model that predicts all classes. Any binary classifier can be used as the basis for a one-versus-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 component. The component would create two-class support vector machine models for all members of the output class. It would then apply the one-versus-all method to combine the results for all classes.

The component uses OneVsRestClassifier of sklearn, and you can learn more details here.

How to configure the One-vs-All Multiclass classifier

This component creates an ensemble of binary classification models to analyze multiple classes. To use this component, you need to configure and train a *binary classification* model first.

You connect the binary model to the One-vs-All Multiclass component. You then train the ensemble of models by using Train Model with a labeled training dataset.

When you combine the models, One-vs-All Multiclass creates multiple binary classification models, optimizes the algorithm for each class, and then merges the models. The component does these tasks even though the training dataset might have multiple class values.

1. Add the One-vs-All Multiclass component to your pipeline in the designer. You can find this component 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's provided as input.

2. Add a binary classification model to the pipeline, and configure that model. For example, you might use

Two-Class Support Vector Machine or Two-Class Boosted Decision Tree.

3. Add the Train Model component to your pipeline. 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 dataset that has multiple class values.
5. Submit the pipeline.

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 dataset.

Next steps

See the set of components available to Azure Machine Learning.

One-vs-One Multiclass

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the One-vs-One Multiclass component in Azure Machine Learning designer. The goal is to create a classification model that can predict multiple classes, by using the *one-versus-one* approach.

This component 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-versus-one models

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). But even binary classification algorithms can be adapted for multi-class classification tasks through a variety of strategies.

This component implements the one-versus-one method, in which a binary model is created per class pair. At prediction time, the class which received the most votes is selected. Since it requires to fit `n_classes * (n_classes - 1) / 2` classifiers, this method is usually slower than one-versus-all, due to its O(n_classes^2) complexity. However, this method may be advantageous for algorithms such as kernel algorithms which don't scale well with `n_samples`. This is because each individual learning problem only involves a small subset of the data whereas, with one-versus-all, the complete dataset is used `n_classes` times.

In essence, the component creates an ensemble of individual models and then merges the results, to create a single model that predicts all classes. Any binary classifier can be used as the basis for a one-versus-one model.

For example, let's say you configure a Two-Class Support Vector Machine model and provide that as input to the One-vs-One Multiclass component. The component would create two-class support vector machine models for all members of the output class. It would then apply the one-versus-one method to combine the results for all classes.

The component uses OneVsOneClassifier of sklearn, and you can learn more details here.

How to configure the One-vs-One Multiclass classifier

This component creates an ensemble of binary classification models to analyze multiple classes. To use this component, you need to configure and train a *binary classification* model first.

You connect the binary model to the One-vs-One Multiclass component. You then train the ensemble of models by using Train Model with a labeled training dataset.

When you combine the models, One-vs-One Multiclass creates multiple binary classification models, optimizes the algorithm for each class, and then merges the models. The component does these tasks even though the training dataset might have multiple class values.

1. Add the One-vs-One Multiclass component to your pipeline in the designer. You can find this component under **Machine Learning - Initialize**, in the **Classification** category.

   The One-vs-One Multiclass classifier has no configurable parameters of its own. Any customizations must be done in the binary classification model that's provided as input.

2. Add a binary classification model to the pipeline, and configure that model. For example, you might use Two-Class Support Vector Machine or Two-Class Boosted Decision Tree.

3. Add the Train Model component to your pipeline. Connect the untrained classifier that is the output of One-vs-One Multiclass.
4. On the other input of Train Model, connect a labeled training dataset that has multiple class values.
5. Submit the pipeline.

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 dataset.

Next steps

See the set of components available to Azure Machine Learning.

Two-Class Averaged Perceptron component

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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. The trained model can then be used to predict values for the new input examples.

About averaged perceptron models

The *averaged perceptron method* is an early and 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** component to your pipeline.
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**: 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. For **Learning rate**, specify a value for the *learning rate*. The learning rate values control 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 pipeline across runs.
6. Connect a training dataset, and train the model:
   - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.

- 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 default value in the single parameter list.
>
> If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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.

Next steps

See the set of components available to Azure Machine Learning.

Two-Class Boosted Decision Tree component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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.

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 large datasets that some linear learners can handle.

This component is based on LightGBM algorithm.

How to configure

This component 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 using Train Model.

1. In Azure Machine Learning, add the **Boosted Decision Tree** component to your pipeline.
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 component. 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 five 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 large, 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.

   If you set the value to 1, 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. Train the model:

   - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
   - 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 default value in the single parameter list.
   >
   > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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 save a snapshot of the trained model, select the **Outputs** tab in the right panel of the **Train model** component. Select the **Register dataset** icon to save the model as a reusable component.
- To use the model for scoring, add the **Score Model** component to a pipeline.

Next steps

See the set of components available to Azure Machine Learning.

Two-Class Decision Forest component

3/28/2022 • 5 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to create a machine learning model based on the decision forests algorithm.

Decision forests are fast, supervised ensemble models. This component is a good choice if you want to predict a target with a maximum of two outcomes.

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.

How to configure

1. Add the **Two-Class Decision Forest** component to your pipeline in Azure Machine Learning, and open the **Properties** pane of the component.

   You can find the component 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 unnormalized 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 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.

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 using the Tune Model Hyperparameters component. 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.

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**
   > If you 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 **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 five cases that meet the same conditions.

7. 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.

8. Attach a labeled dataset, and train the model:

   - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.

   - 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 default value in the single parameter list.

If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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 save a snapshot of the trained model, select the **Outputs** tab in the right panel of the **Train model** component. Select the **Register dataset** icon to save the model as a reusable component.
- To use the model for scoring, add the **Score Model** component to a pipeline.

Next steps

See the set of components available to Azure Machine Learning.

Two-Class Logistic Regression component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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.

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.

In this component, the classification algorithm is optimized for dichotomous or binary variables. if you need to classify multiple outcomes, use the Multiclass Logistic Regression component.

How to configure

To train this model, you must provide a dataset that contains a label or class column. Because this component 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** component to your pipeline.
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 component. 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 **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 pipeline.

7. Add a labeled dataset to the pipeline, and train the model:

   - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
   - 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 default value in the single parameter list.
   >
   > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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. Submit the pipeline.

Results

After training is complete:

- To make predictions on new data, use the trained model and new data as input to the Score Model component.

Next steps

See the set of components available to Azure Machine Learning.

Two-Class Neural Network component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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. 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 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

1. Add the **Two-Class Neural Network** component to your pipeline. You can find this component 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**: If you are not sure of the best parameters, you can find the optimal parameters by using the Tune Model Hyperparameters component. 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 **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.

4. 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.
5. For **Number of learning iterations**, specify the maximum number of times the algorithm should process the training cases.
6. For **The initial learning weights diameter**, specify the node weights at the start of the learning process.
7. For **The momentum**, specify a weight to apply during learning to nodes from previous iterations
8. 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 pipeline.
9. 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 pipeline. Otherwise, a system clock value is used as the seed, which can cause slightly different results each time you run the pipeline.
10. Add a labeled dataset to the pipeline, and train the model:

    - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
    - 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 default value in the single parameter list.
    >
    > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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. Submit the pipeline.

Results

After training is complete:

- To save a snapshot of the trained model, select the **Outputs** tab in the right panel of the **Train model** component. Select the **Register dataset** icon to save the model as a reusable component.
- To use the model for scoring, add the **Score Model** component to a pipeline.

Next steps

See the set of components available to Azure Machine Learning.

Two-Class Support Vector Machine component

3/28/2022 • 3 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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 the training components, and providing a *tagged dataset* that includes a label or outcome column.

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

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** component to your pipeline.
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 component. 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 **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. Connect a labeled dataset, and train the model:

   - If you set **Create trainer mode** to **Single Parameter**, connect a tagged dataset and the Train Model component.
   - 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 default value in the single parameter list.
   >
   > If you pass a single set of parameter values to the Tune Model Hyperparameters component, 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. Submit the pipeline.

Results

After training is complete:

- To save a snapshot of the trained model, select the **Outputs** tab in the right panel of the **Train model** component. Select the **Register dataset** icon to save the model as a reusable component.
- To use the model for scoring, add the **Score Model** component to a pipeline.

Next steps

See the set of components available to Azure Machine Learning.

Train Clustering Model

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to train a clustering model.

The component takes an untrained clustering model that you have already configured using the K-Means Clustering component, and trains the model using a labeled or unlabeled data set. The component 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 component, which is the generic component for training 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** component to your pipeline in the designer. You can find the component under **Machine Learning components**, in the **Train** category.
2. Add the K-Means Clustering component, or another custom component 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. Submit the pipeline, or click the **Train Clustering Model** component and select **Run Selected**.

Results

After training has completed:

- To save a snapshot of the trained model, select the **Outputs** tab in the right panel of the **Train model** component. Select the **Register dataset** icon to save the model as a reusable component.
- To generate scores from the model, use Assign Data to Clusters.

> **NOTE**
> If you need to deploy the trained model in the designer, make sure that Assign Data to Clusters instead of **Score Model** is connected to the input of Web Service Output component in the inference pipeline.

Next steps

See the set of components available to Azure Machine Learning.

Train Model component

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component 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 components 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 components, to make predictions on new data.

How to use Train Model

1. Add the **Train Model** component to the pipeline. You can find this component under the **Machine Learning** category. Expand **Train**, and then drag the **Train Model** component into your pipeline.
2. 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.
3. For **Label column**, click **Edit column** in the right panel of component, 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 component 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.

4. Submit the pipeline. If you have a lot of data, it can take a while.

> **IMPORTANT**
> If you have an ID column which is the ID of each row, or a text column, which contains too many unique values, **Train Model** may hit an error like "Number of unique values in column: "{column_name}" is greater than allowed.
>
> This is because the column hit the threshold of unique values, and may cause out of memory. You can use Edit Metadata to mark that column as **Clear feature** and it will not be used in training, or Extract N-Gram Features from Text component to preprocess text column. See Designer error code for more error details.

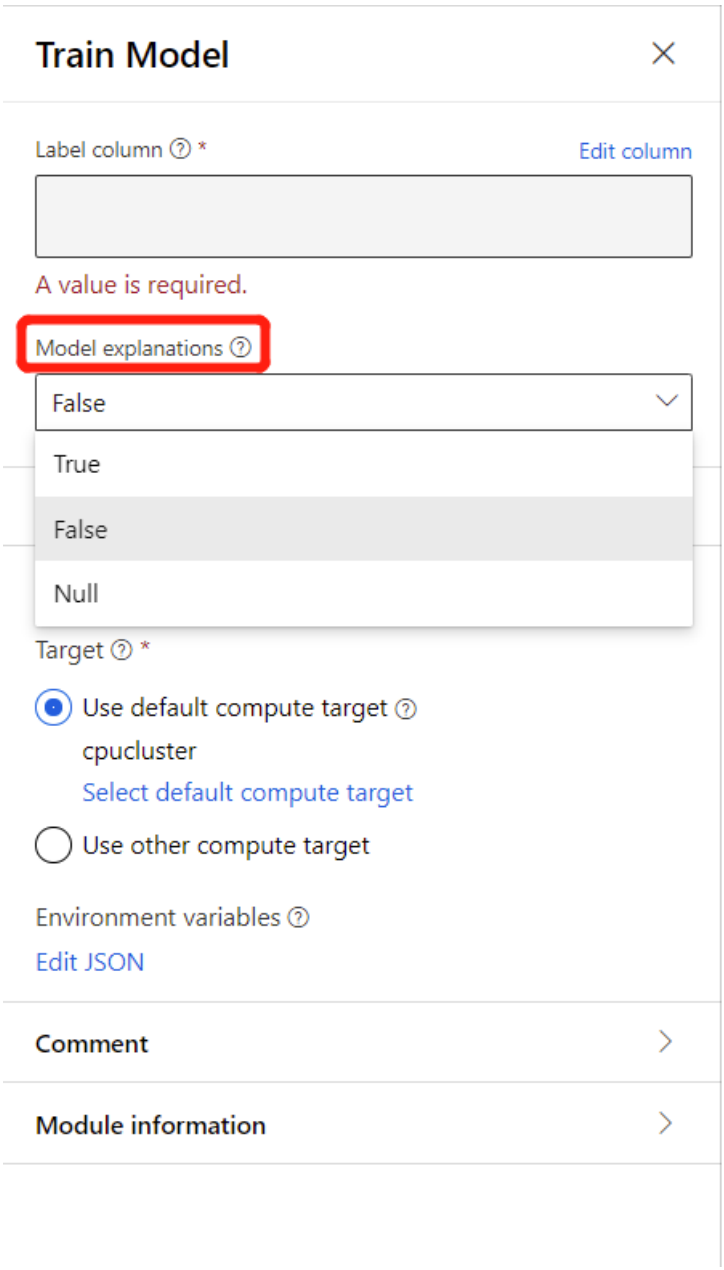
Model Interpretability

Model interpretability provides possibility to comprehend the ML model and to present the underlying basis for decision-making in a way that is understandable to humans.

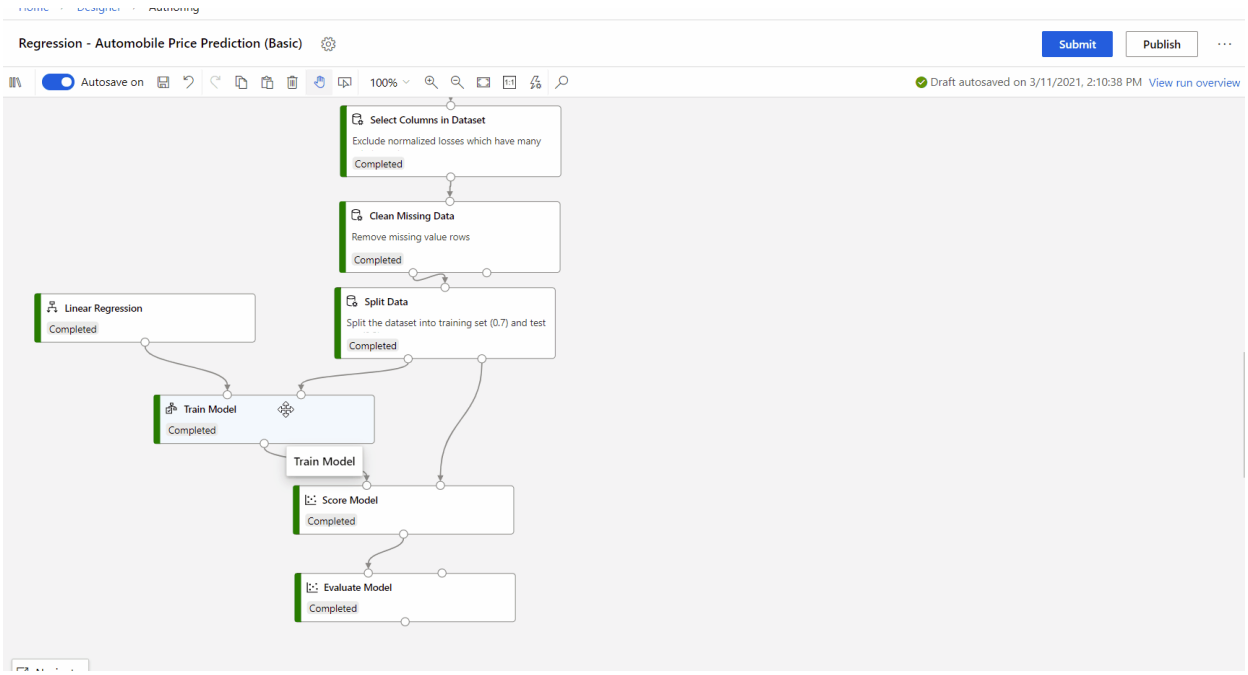
Currently **Train Model** component supports using interpretability package to explain ML models. Following built-in algorithms are supported:

- Linear Regression
- Neural Network Regression
- Boosted Decistion Tree Regression
- Decision Forest Regression
- Poisson Regression
- Two-Class Logistic Regression
- Two-Class Support Vector Machine
- Two-Class Boosted Decistion Tree
- Two-Class Decision Forest
- Multi-class Decision Forest
- Multi-class Logistic Regression
- Multi-class Neural Network

To generate model explanations, you can select **True** in the drop-down list of **Model Explanation** in Train Model component. By default it is set to False in the **Train Model** component. Please note that generating explanation requires extra compute cost.



After the pipeline run completed, you can visit **Explanations** tab in the right pane of **Train Model** component, and explore the model performance, dataset and feature importance.



To learn more about using model explanations in Azure Machine Learning, refer to the how-to article about Interpret ML models.

Results

After the model is trained:

- To use the model in other pipelines, select the component and select the **Register dataset** icon under the **Outputs** tab in right panel. You can access saved models in the component palette under **Datasets**.
- To use the model in predicting new values, connect it to the Score Model component, together with new input data.

Next steps

See the set of components available to Azure Machine Learning.

Train PyTorch Model

3/28/2022 • 5 minutes to read • Edit Online

This article describes how to use the **Train PyTorch Model** component in Azure Machine Learning designer to train PyTorch models like DenseNet. Training takes place after you define a model and set its parameters, and requires labeled data.

Currently, **Train PyTorch Model** component supports both single node and distributed training.

How to use Train PyTorch Model

1. Add DenseNet component or ResNet to your pipeline draft in the designer.
2. Add the **Train PyTorch Model** component to the pipeline. You can find this component under the **Model Training** category. Expand **Train**, and then drag the **Train PyTorch Model** component into your pipeline.

   > **NOTE**
   >
   > **Train PyTorch Model** component is better run on **GPU** type compute for large dataset, otherwise your pipeline will fail. You can select compute for specific component in the right pane of the component by setting **Use other compute target**.

3. On the left input, attach an untrained model. Attach the training dataset and validation dataset to the middle and right-hand input of **Train PyTorch Model**.

   For untrained model, it must be a PyTorch model like DenseNet; otherwise, a 'InvalidModelDirectoryError' will be thrown.

   For dataset, the training dataset must be a labeled image directory. Refer to **Convert to Image Directory** for how to get a labeled image directory. If not labeled, a 'NotLabeledDatasetError' will be thrown.

   The training dataset and validation dataset have the same label categories, otherwise a InvalidDatasetError will be thrown.

4. For **Epochs**, specify how many epochs you'd like to train. The whole dataset will be iterated in every epoch, by default 5.
5. For **Batch size**, specify how many instances to train in a batch, by default 16.
6. For **Warmup step number**, specify how many epochs you'd like to warm up the training, in case initial learning rate is slightly too large to start converging, by default 0.
7. For **Learning rate**, specify a value for the *learning rate*, and the default value is 0.001. Learning rate controls the size of the step that is used in optimizer like sgd each time the model is tested and corrected.

   By setting the rate smaller, you test the model more often, with the risk that you might get stuck in a local plateau. By setting the rate larger, you can converge faster, with the risk of overshooting the true minima.

> **NOTE**
>
> If train loss becomes nan during training, which may be caused by too large learning rate, decreasing learning rate may help. In distributed training, to keep gradient descent stable, the actual learning rate is calculated by `lr * torch.distributed.get_world_size()` because batch size of the process group is world size times that of single process. Polynomial learning rate decay is applied and can help result in a better performing model.

8. For **Random 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.
9. For **Patience**, specify how many epochs to early stop training if validation loss does not decrease consecutively. by default 3.
10. For **Print frequency**, specify training log print frequency over iterations in each epoch, by default 10.
11. Submit the pipeline. If your dataset has larger size, it will take a while and GPU compute are recommended.

Distributed training

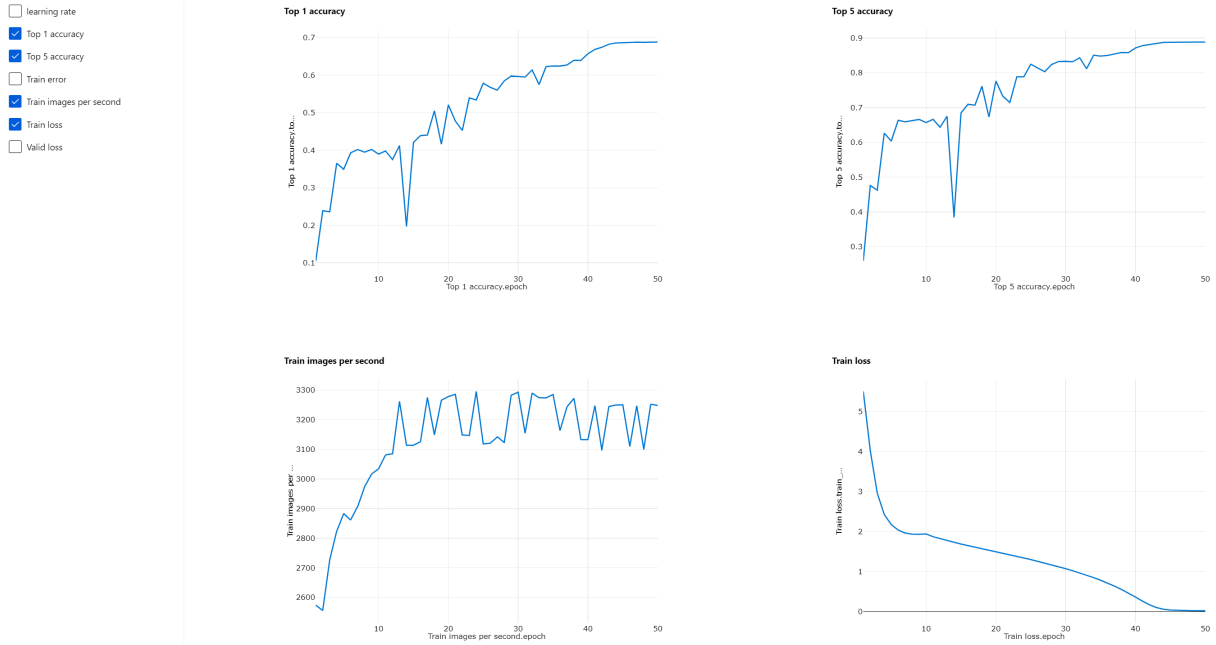
In distributed training the workload to train a model is split up and shared among multiple mini processors, called worker nodes. These worker nodes work in parallel to speed up model training. Currently the designer support distributed training for **Train PyTorch Model** component.

Training time

Distributed training makes it possible to train on a large dataset like ImageNet (1000 classes, 1.2 million images) in just several hours by **Train PyTorch Model**. The following table shows training time and performance during training 50 epochs of Resnet50 on ImageNet from scratch based on different devices.

DEVICES	TRAINING TIME	TRAINING THROUGHPUT	TOP-1 VALIDATION ACCURACY	TOP-5 VALIDATION ACCURACY
16 V100 GPUs	6h22min	~3200 Images/Sec	68.83%	88.84%
8 V100 GPUs	12h21min	~1670 Images/Sec	68.84%	88.74%

Click on this component 'Metrics' tab and see training metric graphs, such as 'Train images per second' and 'Top 1 accuracy'.

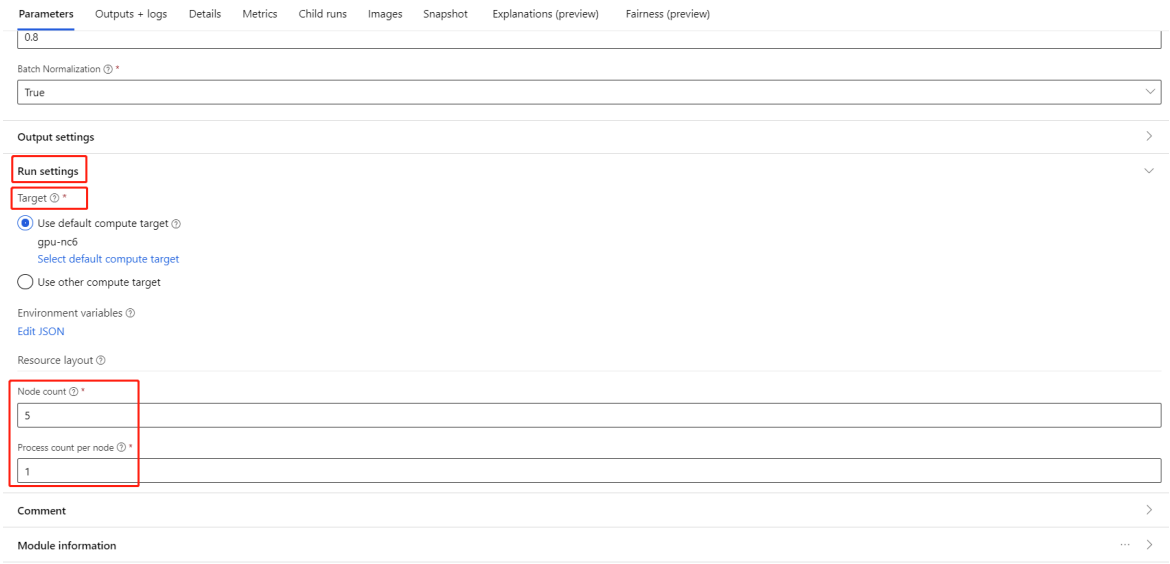


How to enable distributed training

To enable distributed training for **Train PyTorch Model** component, you can set in **Run settings** in the right pane of the component. Only **AML Compute cluster** is supported for distributed training.

> **NOTE**
>
> **Multiple GPUs** are required to activate distributed training because NCCL backend Train PyTorch Model component uses needs cuda.

1. Select the component and open the right panel. Expand the **Run settings** section.



2. Make sure you have select AML compute for the compute target.

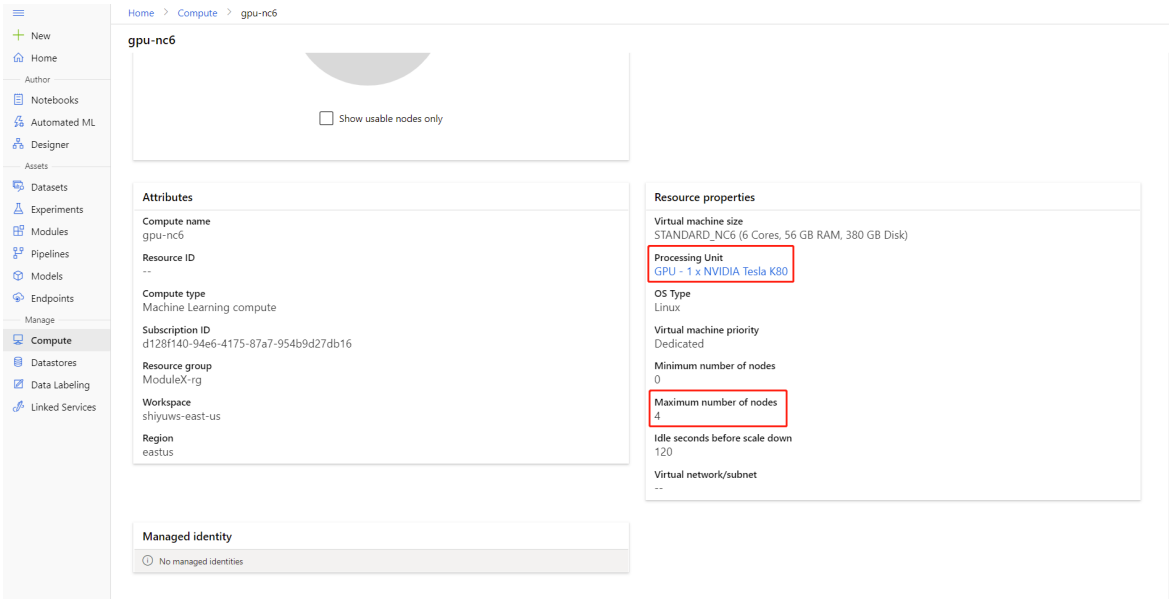
3. In **Resource layout** section, you need to set the following values:

   - **Node count** : Number of nodes in the compute target used for training. It should be **less than or equal to** the **Maximum number of nodes** your compute cluster. By default it is 1, which means single node job.

   - **Process count per node**: Number of processes triggered per node. It should be **less than or equal to** the **Processing Unit** of your compute. By default it is 1, which means single process

job.

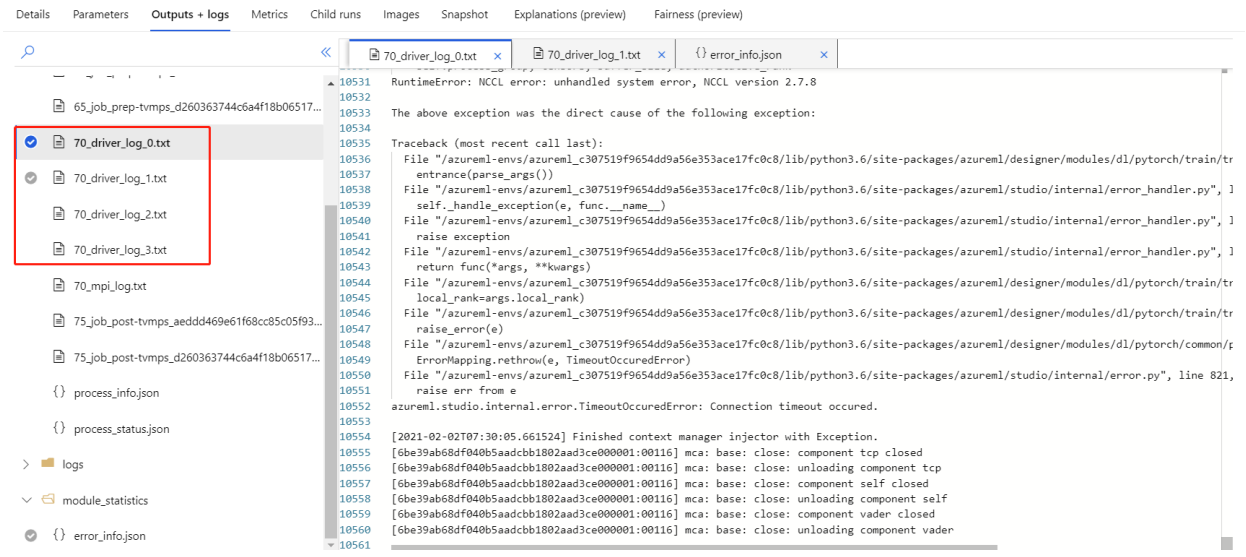
You can check the **Maximum number of nodes** and **Processing Unit** of your compute by clicking the compute name into the compute detail page.



You can learn more about distributed training in Azure Machine Learning here.

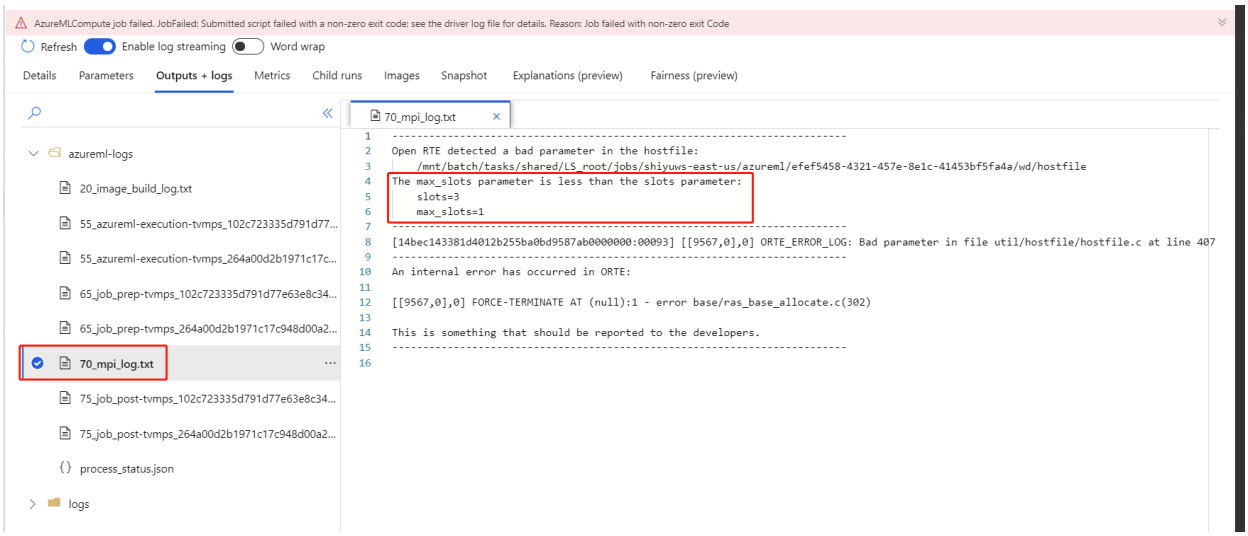
Troubleshooting for distributed training

If you enable distributed training for this component, there will be driver logs for each process. `70_driver_log_0` is for master process. You can check driver logs for error details of each process under **Outputs+logs** tab in the right pane.



If the component enabled distributed training fails without any `70_driver` logs, you can check `70_mpi_log` for error details.

The following example shows a common error, which is **Process count per node** is larger than **Processing Unit** of the compute.



You can refer to this article for more details about component troubleshooting.

Results

After pipeline run is completed, to use the model for scoring, connect the Train PyTorch Model to Score Image Model, to predict values for new input examples.

Technical notes

Expected inputs

NAME	TYPE	DESCRIPTION
Untrained model	UntrainedModelDirectory	Untrained model, require PyTorch
Training dataset	ImageDirectory	Training dataset
Validation dataset	ImageDirectory	Validation dataset for evaluation every epoch

Component parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Epochs	>0	Integer	5	Select the column that contains the label or outcome column
Batch size	>0	Integer	16	How many instances to train in a batch
Warmup step number	>=0	Integer	0	How many epochs to warm up training
Learning rate	>=double.Epsilon	Float	0.1	The initial learning rate for the Stochastic Gradient Descent optimizer.

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Random seed	Any	Integer	1	The seed for the random number generator used by the model.
Patience	>0	Integer	3	How many epochs to early stop training
Print frequency	>0	Integer	10	Training log print frequency over iterations in each epoch

Outputs

NAME	TYPE	DESCRIPTION
Trained model	ModelDirectory	Trained model

Next steps

See the set of components available to Azure Machine Learning.

Tune Model Hyperparameters

3/28/2022 • 7 minutes to read • Edit Online

This article describes how to use the Tune Model Hyperparameters component in Azure Machine Learning designer. The goal is to determine the optimum hyperparameters for a machine learning model. The component builds and tests multiple models by using different combinations of settings. It compares metrics over all models to get the combinations of settings.

The terms *parameter* and *hyperparameter* can be confusing. The model's *parameters* are what you set in the right pane of the component. Basically, this component performs a *parameter sweep* over the specified parameter settings. It 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 component supports the following method for finding the optimum settings for a model: *integrated train and tune*. In this method, you configure a set of parameters to use. You then let the component iterate over multiple combinations. The component measures accuracy until it finds a "best" model. With most learner components, 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 might shorten the process by establishing a grid of parameter combinations and testing a randomized subset of the parameter grid.

This method generates a trained model that you can save for reuse.

> **TIP**
> You can do a related task. Before you start tuning, apply feature selection to determine the columns or variables that have the highest information value.

How to configure Tune Model Hyperparameters

Learning the optimal hyperparameters for a machine learning model requires considerable use of pipelines.

Train a model by using a parameter sweep

This section describes how to perform a basic parameter sweep, which trains a model by using the Tune Model Hyperparameters component.

1. Add the Tune Model Hyperparameters component to your pipeline in the designer.
2. Connect an untrained model to the leftmost input.

   > **NOTE**
   > **Tune Model Hyperparameters** can only be connect to built-in machine learning algorithm components, and cannot support customized model built in **Create Python Model**.

3. Add the dataset that 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.

4. In the right panel 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 component loops over a grid predefined by the system, to try different combinations and identify the best learner. This option is useful when you don't know what the best parameter settings might be and want to try all possible combinations of values.

   - **Random sweep**: When you select this option, the component will randomly select parameter values over a system-defined range. You must specify the maximum number of runs that you want the component to execute. This option is useful when you want to increase model performance by using the metrics of your choice but still conserve computing resources.

5. For **Label column**, open the column selector to choose a single label column.

6. Choose the number of runs:

   - **Maximum number of runs on random sweep**: If you choose a random sweep, you can specify how many times the model should be trained, by using a random combination of parameter values.

7. For **Ranking**, choose a single metric to use for ranking the models.

   When you run a parameter sweep, the component calculates all applicable metrics for the model type and returns them in the **Sweep results** report. The component uses separate metrics for regression and classification models.

   However, the metric that 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**, enter an integer number as a pseudo random number generator state used for randomly selecting parameter values over a pre-defined range. This parameter is only effective if **Parameter sweeping mode** is **Random sweep**.

9. Submit the pipeline.

Results of hyperparameter tuning

When training is complete:

- To view the sweep results, you could either right-click the component, and then select **Visualize**, or right-click left output port of the component to visualize.

  The **Sweep results** includes all parameter sweep and accuracy metrics that apply to the model type, and the metric that you selected for ranking determines which model is considered "best."

- To save a snapshot of the trained model, select the **Outputs+logs** tab in the right panel of the **Train model** component. Select the **Register dataset** icon to save the model as a reusable component.

Technical notes

This section contains implementation details and tips.

How a parameter sweep works

When you set up a parameter sweep, you define the scope of your search. The search might use a finite number of parameters selected randomly. Or it might be an exhaustive search over a parameter space that you define.

- **Random sweep**: This option trains a model by using a set number of iterations.

You specify a range of values to iterate over, and the component 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. So the chance of any value being selected stays the same across all passes.

- **Entire grid**: The option to use the entire grid means that every combination is tested. This option is the most thorough, but it requires the most time.

Controlling the length and complexity of training

Iterating over many combinations of settings can be time-consuming, so the component 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 pipeline with the settings to determine the most efficient method of training on a particular dataset and model.

Choosing an evaluation metric

At the end of testing, the model presents a report that contains the accuracy for each model so that you can review the metric results:

- A uniform set of metrics is used for all binary classification models.
- Accuracy is used for all multi-class classification models.
- 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.

Metrics used for binary classification

- **Accuracy** is the proportion of true results to total cases.
- **Precision** is the proportion of true results to positive results.
- **Recall** is the fraction of all correct results over all results.
- **F-score** is a measure that balances precision and recall.
- **AUC** is 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** is the difference between two probability distributions: the true one, and the one in the model.

Metrics used for regression

- **Mean absolute error** averages all the errors in the model, where *error* means the distance of the predicted value from the true value. It's 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. It's 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** is a single number that indicates how well data fits a model. A value of one means that the model exactly matches the data. A value of zero means that the data is random or otherwise can't be fit to the model. It's often called r^2, R^2, or *r-squared*.

Components that don't 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 pipeline with. If the learner doesn't support setting a range of values, you can still use it in cross-validation. In this case, a range of allowed values is selected for the sweep.

Next steps

See the set of components available to Azure Machine Learning.

Apply Transformation component

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to modify an input dataset based on a previously computed transformation. This component is necessary in if you need to update transformations in inference pipelines.

For example, if you used z-scores to normalize your training data by using the **Normalize Data** component, you would want to use the z-score value that was computed for training during the scoring phase as well. In Azure Machine Learning, you can save the normalization method as a transform, and then using **Apply Transformation** to apply the z-score to the input data before scoring.

How to save transformations

The designer lets you save data transformations as **datasets** so that you can use them in other pipelines.

1. Select a data transformation component that has successfully run.
2. Select the **Outputs + logs** tab.
3. Find the transformation output, and select the **Register dataset** to save it as a component under **Datasets** category in the component palette.

How to use Apply Transformation

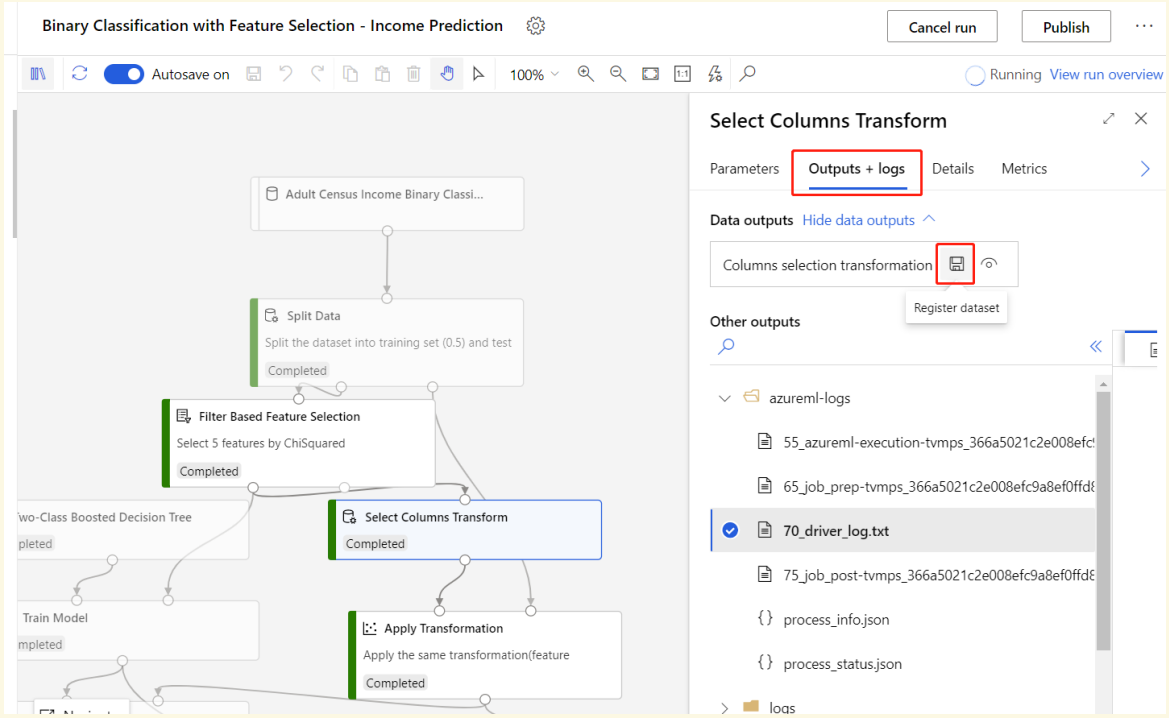
1. Add the **Apply Transformation** component to your pipeline. You can find this component in the **Model Scoring & Evaluation** section of the component palette.
2. Find the saved transformation you want to use under **Datasets** in the component palette.
3. Connect the output of the saved transformation to the left input port of the **Apply Transformation** component.

   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. Connect the dataset output of the desired component to the right input port of the **Apply Transformation** component.
5. To apply a transformation to the new dataset, submit the pipeline.

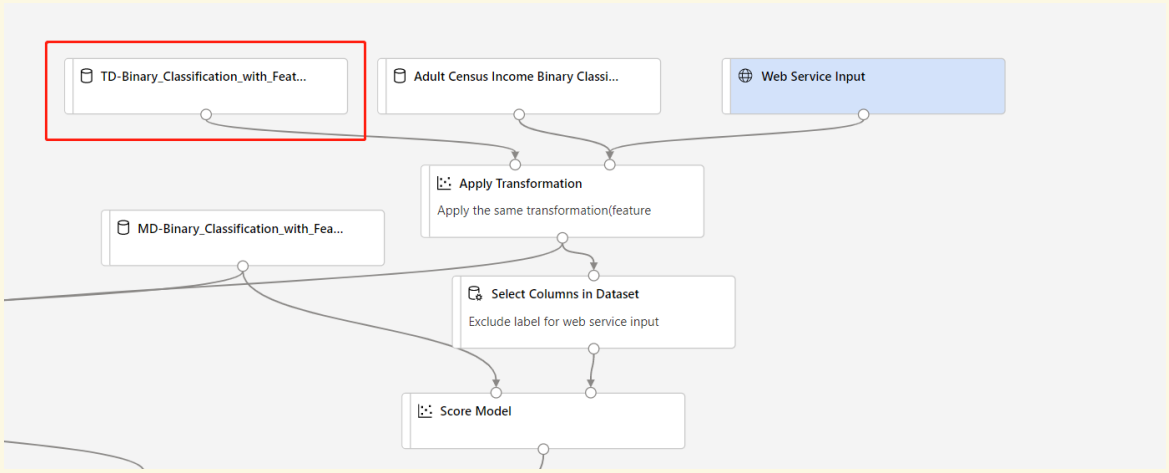
IMPORTANT

To make sure the updated transformation in training pipelines is also feasible in inference pipelines, you need to follow the steps below each time there is updated transformation in the training pipeline:

1. In the training pipeline, register the output of the Select Columns Transform as a dataset.



2. In the inference pipeline, remove the **TD-** component, and replace it with the registered dataset in the previous step.



Next steps

See the set of components available to Azure Machine Learning.

Component: Assign Data to Clusters

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the *Assign Data to Clusters* component in Azure Machine Learning designer. The component generates predictions through a clustering model that was trained with the *K-means clustering* algorithm.

The Assign Data to Clusters component returns a dataset that contains the probable assignments for each new data point.

How to use Assign Data to Clusters

1. In Azure Machine Learning designer, locate a previously trained clustering model. You can create and train a clustering model by using either of the following methods:

   - Configure the K-means clustering algorithm by using the K-Means Clustering component, and train the model by using a dataset and the Train Clustering Model component (this article).
   - 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. You are not expected to know the 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 that are written to the designer from the cluster predictions, use Select columns in the dataset, and select a subset of the columns.

4. Leave the **Check for append or uncheck for result only** check box selected if you want the results to contain the full input dataset, including a column that displays the results (cluster assignments).

   If you clear this check box, only the results are returned. This option might be useful when you create predictions as part of a web service.

5. Submit the pipeline.

Results

- To view the values in the dataset, right-click the component, and then select **Visualize**. Or Select the component and switch to the **Outputs** tab in the right panel, click on the histogram icon in the **Port outputs** to visualize the result.

Cross Validate Model

3/28/2022 • 6 minutes to read • Edit Online

This article describes how to use the Cross Validate Model component in Azure Machine Learning designer. *Cross-validation* is a technique often used in machine learning to assess both the variability of a dataset and the reliability of any model trained through that data.

The Cross Validate Model component 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. You can then understand whether the model is susceptible to variations in the data.

Cross Validate Model 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 training data into 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 component and indicate how many folds to use.

2. The component sets aside the data in fold 1 to use for validation. (This is sometimes called the *holdout fold*.) The component uses the remaining folds to train a model.

   For example, if you create five folds, the component generates five models during cross-validation. The component trains each model by using four-fifths of the data. It tests each model on the remaining one-fifth.

3. During testing of the model for each fold, the component evaluates multiple accuracy statistics. Which statistics the component uses depends on the type of model that you're evaluating. Different statistics are used to evaluate classification models versus 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. Review these metrics to see whether any single fold has high or low accuracy.

Advantages of cross-validation

A different and common way of evaluating a model is to divide the data into a training and test set by using Split Data, and then validate the model on the training data. But 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 a portion. In contrast, if you validate a model by using data generated from a random split, typically you evaluate the model on only 30 percent or less of the available data.

  However, because cross-validation trains and validates the model multiple times over a larger dataset, it's much more computationally intensive. It takes much longer than validating on a random split.

- Cross-validation evaluates both the dataset and the model.

Cross-validation doesn't simply measure the accuracy of a model. It 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

Cross-validation can take a long time to run if your dataset is large. So, you might use Cross Validate Model in the initial phase of building and testing your model. In that phase, you can evaluate the goodness of the model parameters (assuming that computation time is tolerable). You can then train and evaluate your model by using the established parameters with the Train Model and Evaluate Model components.

In this scenario, you both train and test the model by using Cross Validate Model.

1. Add the Cross Validate Model component to your pipeline. You can find it in Azure Machine Learning designer, in the **Model Scoring & Evaluation** category.
2. Connect the output of any classification or regression model.

   For example, if you're using **Two Class Boosted Decision Tree** for classification, configure the model with the parameters that you want. Then, drag a connector from the **Untrained model** port of the classifier to the matching port of Cross Validate Model.

   > **TIP**
   > You don't have to train the model, 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 right panel of Cross Validate Model, click **Edit column**. Select 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 repeat the results of cross-validation across successive runs on the same data.
6. Submit the pipeline.
7. See the Results section for a description of the reports.

Results

After all iterations are complete, Cross Validate Model creates scores for the entire dataset. It also creates performance metrics that you can use to assess the quality of the model.

Scored results

The first output of the component provides the source data for each row, together with some predicted values and related probabilities.

To view the results, in the pipeline, right-click the Cross Validate Model component. Select **Visualize Scored results**.

NEW COLUMN NAME	DESCRIPTION
Scored Labels	This column is added at the end of the dataset. It contains the predicted value for each row.
Scored Probabilities	This column is added at the end of the dataset. It indicates the estimated probability of the value in **Scored Labels**.

NEW COLUMN NAME	DESCRIPTION
Fold Number	Indicates the zero-based index of the fold that each row of data was assigned to during cross-validation.

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. It 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 the results, in the pipeline, right-click the Cross Validate Model component. Select **Visualize Evaluation results by fold**.

COLUMN NAME	DESCRIPTION
Fold number	An identifier for each fold. If you created five folds, there would be five 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.

The component also includes the following metrics for each fold, depending on the type of model that you're evaluating:

- **Classification models**: Precision, recall, F-score, AUC, accuracy
- **Regression models**: Mean absolute error, root mean squared error, relative absolute error, relative squared error, and coefficient of determination

Technical notes

- It's a best practice to normalize datasets before you use them for cross-validation.
- Cross Validate Model is much more computationally intensive and takes longer to complete than if you validated the model by using a randomly divided dataset. The reason is that Cross Validate Model trains and validates the model multiple times.
- There's no need to split the dataset into training and testing sets when you use cross-validation to measure the accuracy of the model.

Next steps

See the set of components available to Azure Machine Learning.

Evaluate Model component

3/28/2022 • 6 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to measure the accuracy of a trained model. You provide a dataset containing scores generated from a model, and the **Evaluate Model** component 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**

> **TIP**
> If you are new to model evaluation, we recommend the video series by Dr. Stephen Elston, as part of the machine learning course from EdX.

How to use Evaluate Model

1. Connect the **Scored dataset** output of the Score Model or Result dataset output of the Assign Data to Clusters to the left input port of **Evaluate Model**.

   > **NOTE**
   > If use components like "Select Columns in Dataset" to select part of input dataset, please ensure Actual label column (used in training), 'Scored Probabilities' column and 'Scored Labels' column exist to calculate metrics like AUC, Accuracy for binary classification/anomaly detection. Actual label column, 'Scored Labels' column exist to calculate metrics for multi-class classification/regression. 'Assignments' column, columns 'DistancesToClusterCenter no.X' (X is centroid index, ranging from 0, ..., Number of centroids-1) exist to calculate metrics for clustering.

   > **IMPORTANT**
   > - To evaluate the results, the output dataset should contain specific score column names, which meet Evaluate Model component requirements.
   > - The `Labels` column will be considered as actual labels.
   > - For regression task, the dataset to evaluate must has one column, named `Regression Scored Labels`, which represents scored labels.
   > - For binary classification task, the dataset to evaluate must has two columns, named `Binary Class Scored Labels`, `Binary Class Scored Probabilities`, which represent scored labels, and probabilities respectively.
   > - For multi classification task, the dataset to evaluate must has one column, named `Multi Class Scored Labels`, which represents scored labels. If the outputs of the upstream component does not have these columns, you need to modify according to the requirements above.

2. [Optional] Connect the **Scored dataset** output of the Score Model or Result dataset output of the Assign Data to Clusters for the second model to the **right** input port of **Evaluate Model**. You can easily

compare results from two different models on the same data. The two input algorithms should be the same algorithm type. Or, you might compare scores from two different runs over the same data with different parameters.

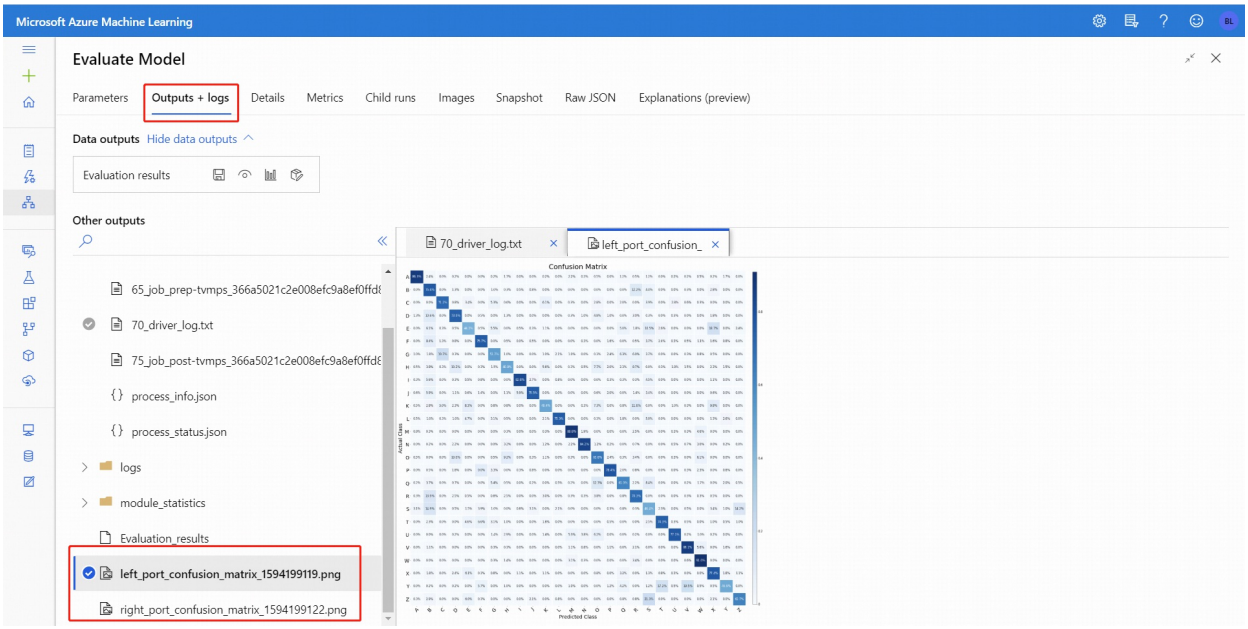
> **NOTE**
>
> Algorithm type refers to 'Two-class Classification', 'Multi-class Classification', 'Regression', 'Clustering' under 'Machine Learning Algorithms'.

3. Submit the pipeline to generate the evaluation scores.

Results

After you run **Evaluate Model**, select the component to open up the **Evaluate Model** navigation panel on the right. Then, choose the **Outputs + Logs** tab, and on that tab the **Data Outputs** section has several icons. The **Visualize** icon has a bar graph icon, and is a first way to see the results.

For binary-classification, after you click **Visualize** icon, you can visualize the binary confusion matrix. For multi-classification, you can find the confusion matrix plot file under the **Outputs + Logs** tab like following:



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 Other Center	Average Distance to Cluster Center	Number of Points	Maximal Distance to Cluster Center
Evaluation For Cluster No.0	0.234019	0.066162	46	0.202332
Evaluation For Cluster No.1	0.244288	0.045689	75	0.116868
Evaluation For Cluster No.2	0.474898	0.085925	14	0.19018
Combined Evaluation	0.264704	0.056838	135	0.202332
Evaluation For Cluster No.0	0.52613	0.16321	22	0.311153
Evaluation For Cluster No.1	0.52613	0.099904	113	0.240107
Combined Evaluation	0.52613	0.11022	135	0.311153

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 binary classification models.

- **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. Precision = TP/(TP+FP)
- **Recall** is the fraction of the total amount of relevant instances that were actually retrieved. Recall = TP/(TP+FN)
- **F1 score** is computed as the weighted average of precision and recall between 0 and 1, where the ideal F1 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. AUC is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

Metrics for regression models

The metrics returned for regression models are 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.

- **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.
- **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.

The following metrics are reported for evaluating clustering models.

- 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.
- 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 **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 max 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.
- The **Combined Evaluation** score at the bottom of each section of results lists the averaged scores for the clusters created in that particular model.

Next steps

See the set of components available to Azure Machine Learning.

Score Image Model

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to generate predictions using a trained image model on input image data.

How to configure Score Image Model

1. Add the **Score Image Model** component to your pipeline.
2. Attach a trained image model and a dataset containing input image data.

   The data should be of type ImageDirectory. Refer to Convert to Image Directory component for more information about how to get a image directory. The schema of the input dataset should also generally match the schema of the data used to train the model.
3. Submit the pipeline.

Results

After you have generated a set of scores using Score Image Model, to generate a set of metrics used for evaluating the model's accuracy (performance), you can connect this component and the scored dataset to Evaluate Model,

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 deploy a real-time endpoint based on a pipeline in Azure Machine Learning designer.

Next steps

See the set of components available to Azure Machine Learning.

Score Model

3/28/2022 • 2 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use this component to generate predictions using a trained classification or regression model.

How to use

1. Add the **Score Model** component to your pipeline.
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. Submit the pipeline.

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 component and select **Visualize** to see a sample of the results.

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.

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 deploy a real-time endpoint based on a pipeline in Azure Machine Learning designer.

Next steps

See the set of components available to Azure Machine Learning.

Create Python Model component

3/28/2022 • 3 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Learn how to use the Create Python Model component to create an untrained model from a Python script. You can base the model on any learner that's included in a Python package in the Azure Machine Learning designer 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 make predictions. You can then save the trained model and publish the scoring workflow as a web service.

WARNING

Currently, it's not possible to connect this component to **Tune Model Hyperparameters** component or pass the scored results of a Python model to Evaluate Model. If you need to tune the hyperparameters or evaluate a model, you can write a custom Python script by using Execute Python Script component.

Configure the component

Use of this component requires intermediate or expert knowledge of Python. The component supports use of any learner that's included in the Python packages already installed in Azure Machine Learning. See the preinstalled Python package list in Execute Python Script.

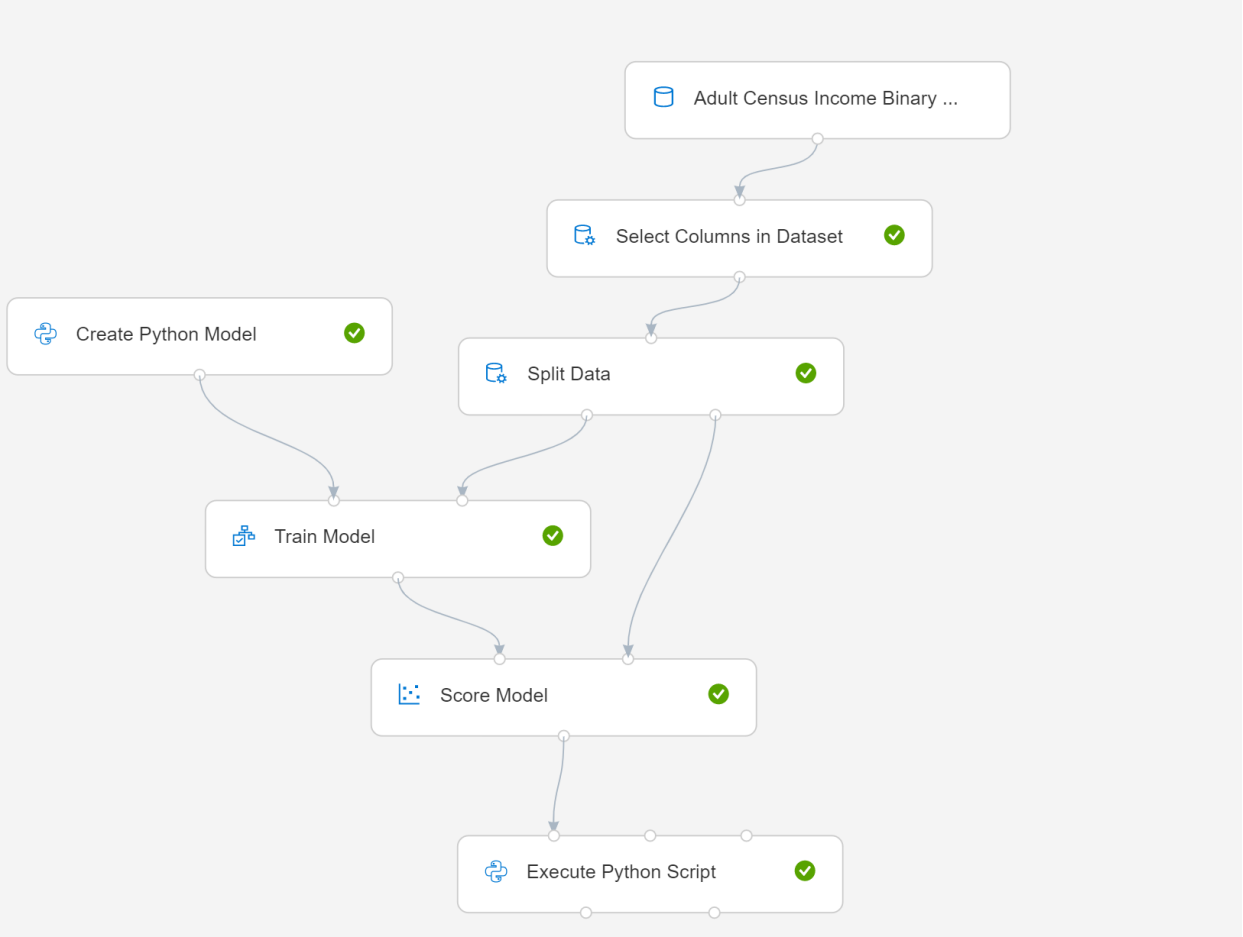
NOTE

Please be very careful when writing your script and makes sure there is no syntax error, such as using a un-declared object or a un-imported component.

NOTE

Also pay extra attentions to the pre-installed components list in Execute Python Script. Only import pre-installed components. Please do not install extra packages such as "pip install xgboost" in this script, otherwise errors will be raised when reading models in down-stream components.

This article shows how to use **Create Python Model** with a simple pipeline. Here's a diagram of the pipeline:



1. Select **Create Python Model**, and edit the script to implement your modeling or data management process. You can base the model on any learner that's included in a Python package in the Azure Machine Learning environment.

> **NOTE**
> Please pay extra attention to the comments in sample code of the script and make sure your script strictly follows the requirement, including the class name, methods as well as method signature. Violation will lead to exceptions. **Create Python Model** only supports creating sklearn based model to be trained using **Train Model**.

The following sample code of the two-class Naive Bayes classifier uses the popular *sklearn* package:

```
# The script MUST define a class named AzureMLModel.
# This class MUST at least define the following three methods:
    # __init__: in which self.model must be assigned,
    # train: which trains self.model, the two input arguments must be pandas DataFrame,
    # predict: which generates prediction result, the input argument and the prediction result MUST be
pandas DataFrame.
# The signatures (method names and argument names) of all these methods MUST be exactly the same as the
following example.

# Please do not install extra packages such as "pip install xgboost" in this script,
# otherwise errors will be raised when reading models in down-stream components.

import pandas as pd
from sklearn.naive_bayes import GaussianNB


class AzureMLModel:
    def __init__(self):
        self.model = GaussianNB()
        self.feature_column_names = list()

    def train(self, df_train, df_label):
        # self.feature_column_names records the column names used for training.
        # It is recommended to set this attribute before training so that the
        # feature columns used in predict and train methods have the same names.
        self.feature_column_names = df_train.columns.tolist()
        self.model.fit(df_train, df_label)

    def predict(self, df):
        # The feature columns used for prediction MUST have the same names as the ones for training.
        # The name of score column ("Scored Labels" in this case) MUST be different from any other columns
in input data.
        return pd.DataFrame(
            {'Scored Labels': self.model.predict(df[self.feature_column_names]),
             'probabilities': self.model.predict_proba(df[self.feature_column_names])[:, 1]}
        )
```

2. Connect the **Create Python Model** component that you just created to **Train Model** and **Score Model**.

3. If you need to evaluate the model, add an Execute Python Script component and edit the Python script.

   The following script is sample evaluation code:

```
# The script MUST contain a function named azureml_main
# which is the entry point for this component.

# imports up here can be used to
import pandas as pd

# The entry point function MUST have two input arguments:
#   Param<dataframe1>: a pandas.DataFrame
#   Param<dataframe2>: a pandas.DataFrame
def azureml_main(dataframe1 = None, dataframe2 = None):

    from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score,
roc_curve
    import pandas as pd
    import numpy as np

    scores = dataframe1.ix[:, ("income", "Scored Labels", "probabilities")]
    ytrue = np.array([0 if val == '<=50K' else 1 for val in scores["income"]])
    ypred = np.array([0 if val == '<=50K' else 1 for val in scores["Scored Labels"]])
    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]

    return metrics,
```

Next steps

See the set of components available to Azure Machine Learning.

Execute Python Script component

3/28/2022 • 8 minutes to read • Edit Online

This article describes the Execute Python Script component in Azure Machine Learning designer.

Use this component to run Python code. For more information about the architecture and design principles of Python, see how run Python code in Azure Machine Learning designer.

With Python, you can perform tasks that existing components don't support, such as:

- Visualizing data by using `matplotlib`.
- Using Python libraries to enumerate datasets and models in your workspace.
- Reading, loading, and manipulating data from sources that the Import Data component doesn't support.
- Run your own deep learning code.

Supported Python packages

Azure Machine Learning uses the Anaconda distribution of Python, which includes many common utilities for data processing. We will update the Anaconda version automatically. The current version is:

- Anaconda 4.5+ distribution for Python 3.6

For a complete list, see the section Preinstalled Python packages.

To install packages that aren't in the preinstalled list (for example, *scikit-misc*), add the following code to your script:

```
import os
os.system(f"pip install scikit-misc")
```

Use the following code to install packages for better performance, especially for inference:

```
import importlib.util
package_name = 'scikit-misc'
spec = importlib.util.find_spec(package_name)
if spec is None:
    import os
    os.system(f"pip install scikit-misc")
```

NOTE

If your pipeline contains multiple Execute Python Script components that need packages that aren't in the preinstalled list, install the packages in each component.

WARNING

Excute Python Script component does not support installing packages that depend on extra native libraries with command like "apt-get", such as Java, PyODBC and etc. This is because this component is executed in a simple environment with Python pre-installed only and with non-admin permission.

Access to current workspace and registered datasets

You can refer to the following sample code to access to the registered datasets in your workspace:

```
def azureml_main(dataframe1 = None, dataframe2 = None):

    # Execution logic goes here
    print(f'Input pandas.DataFrame #1: {dataframe1}')
    from azureml.core import Run
    run = Run.get_context(allow_offline=True)
    #access to current workspace
    ws = run.experiment.workspace

    #access to registered dataset of current workspace
    from azureml.core import Dataset
    dataset = Dataset.get_by_name(ws, name='test-register-tabular-in-designer')
    dataframe1 = dataset.to_pandas_dataframe()

    # If a zip file is connected to the third input port,
    # it is unzipped under "./Script Bundle". This directory is added
    # to sys.path. Therefore, if your zip file contains a Python file
    # mymodule.py you can import it using:
    # import mymodule

    # Return value must be of a sequence of pandas.DataFrame
    # E.g.
    #   -  Single return value: return dataframe1,
    #   -  Two return values: return dataframe1, dataframe2
    return dataframe1,
```

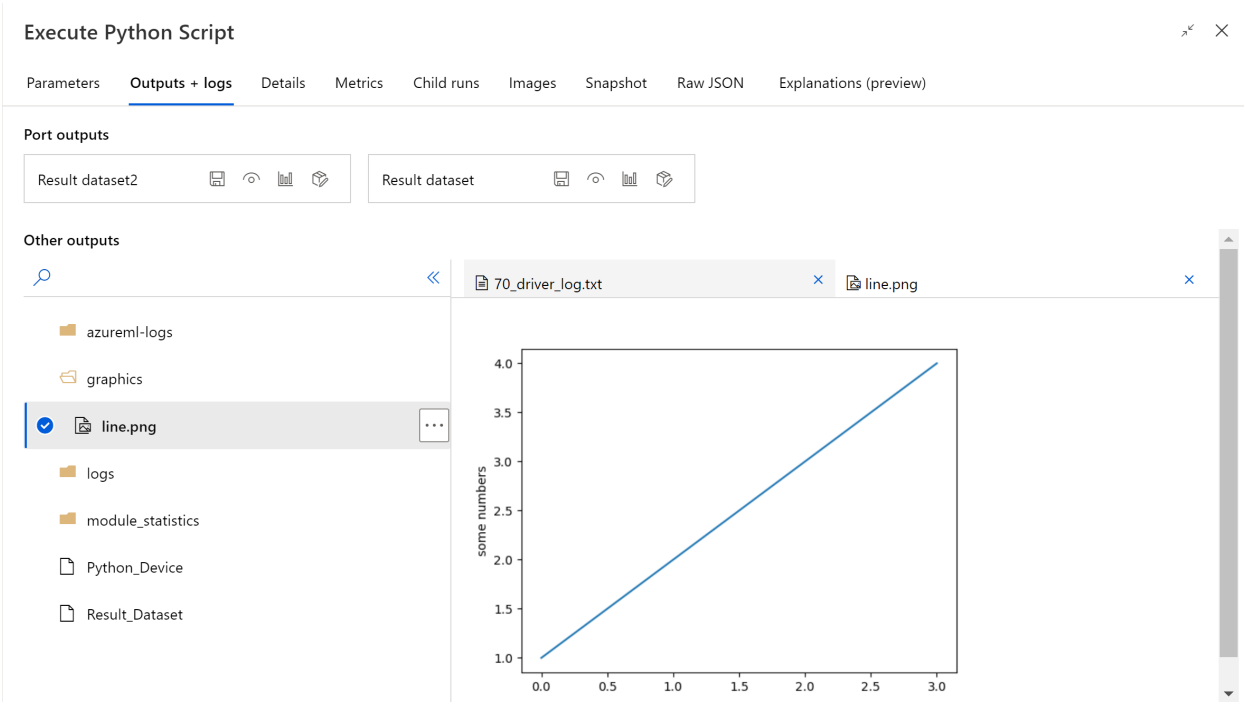
Upload files

The Execute Python Script component supports uploading files by using the Azure Machine Learning Python SDK.

The following example shows how to upload an image file in the Execute Python Script component:

```
# The script MUST contain a function named azureml_main,
# which is the entry point for this component.

# Imports up here can be used to
import pandas as pd

# The entry point function must have two input arguments:
#   Param<dataframe1>: a pandas.DataFrame
#   Param<dataframe2>: a pandas.DataFrame
def azureml_main(dataframe1 = None, dataframe2 = None):

    # Execution logic goes here
    print(f'Input pandas.DataFrame #1: {dataframe1}')

    from matplotlib import pyplot as plt
    plt.plot([1, 2, 3, 4])
    plt.ylabel('some numbers')
    img_file = "line.png"
    plt.savefig(img_file)

    from azureml.core import Run
    run = Run.get_context(allow_offline=True)
    run.upload_file(f"graphics/{img_file}", img_file)

    # Return value must be of a sequence of pandas.DataFrame
    # For example:
    #   -  Single return value: return dataframe1,
    #   -  Two return values: return dataframe1, dataframe2
    return dataframe1,
```

After the pipeline run is finished, you can preview the image in the right panel of the component.



You can also upload file to any datastore using following code. You can only preview the file in your storage account.

```
import pandas as pd

# The entry point function MUST have two input arguments.
# If the input port is not connected, the corresponding
# dataframe argument will be None.
#   Param<dataframe1>: a pandas.DataFrame
#   Param<dataframe2>: a pandas.DataFrame
def azureml_main(dataframe1 = None, dataframe2 = None):

    # Execution logic goes here
    print(f'Input pandas.DataFrame #1: {dataframe1}')

    from matplotlib import pyplot as plt
    import os

    plt.plot([1, 2, 3, 4])
    plt.ylabel('some numbers')
    img_file = "line.png"

    # Set path
    path = "./img_folder"
    os.mkdir(path)
    plt.savefig(os.path.join(path,img_file))

    # Get current workspace
    from azureml.core import Run
    run = Run.get_context(allow_offline=True)
    ws = run.experiment.workspace

    # Get a named datastore from the current workspace and upload to specified path
    from azureml.core import Datastore
    datastore = Datastore.get(ws, datastore_name='workspacefilestore')
    datastore.upload(path)

    return dataframe1,
```

How to configure Execute Python Script

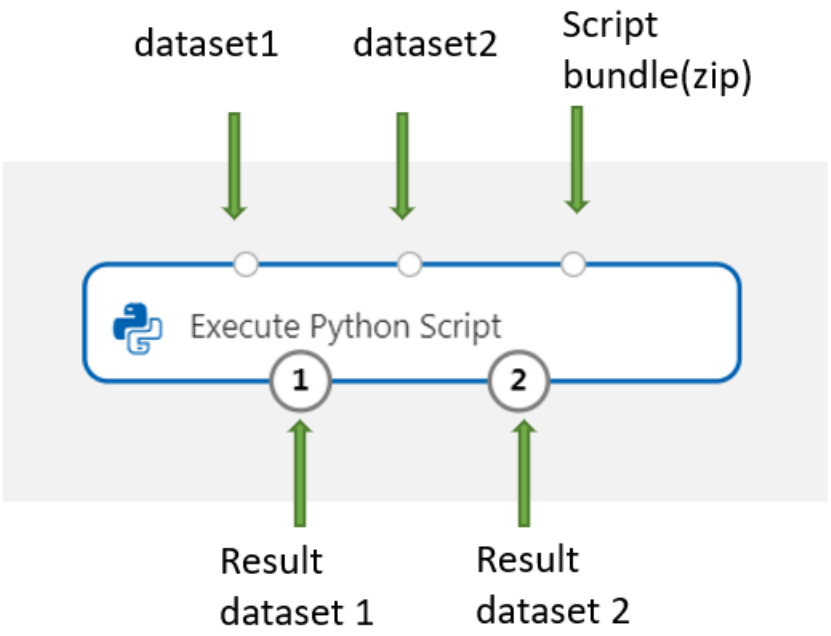
The Execute Python Script component contains sample Python code that you can use as a starting point. To configure the Execute Python Script component, provide a set of inputs and Python code to run in the **Python script** text box.

1. Add the **Execute Python Script** component to your pipeline.
2. Add and connect on **Dataset1** any datasets from the designer that you want to use for input. Reference this dataset in your Python script as **DataFrame1**.

   Use of a dataset is optional. Use it if you want to generate data by using Python, or use Python code to import the data directly into the component.

   This component supports the addition of a second dataset on **Dataset2**. Reference the second dataset in your Python script as **DataFrame2**.

   Datasets stored in Azure Machine Learning are automatically converted to pandas data frames when loaded with this component.



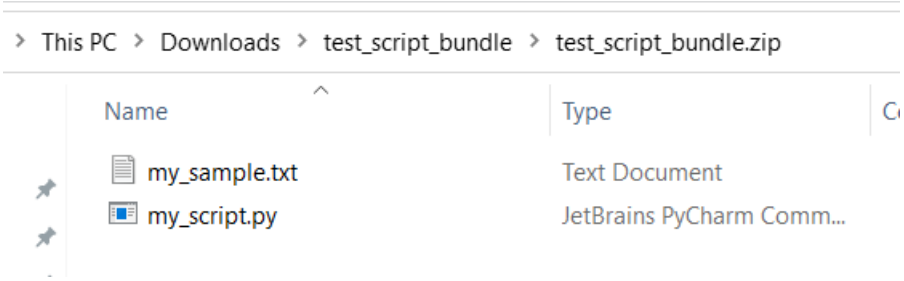
3. To include new Python packages or code, connect the zipped file that contains these custom resources to **Script bundle** port. Or if your script is larger than 16 KB, use the **Script Bundle** port to avoid errors like *CommandLine exceeds the limit of 16597 characters*.

   a. Bundle the script and other custom resources to a zip file.
   b. Upload the zip file as a **File Dataset** to the studio.
   c. Drag the dataset component from the *Datasets* list in the left component pane in the designer authoring page.
   d. Connect the dataset component to the **Script Bundle** port of **Execute Python Script** component.

   Any file contained in the uploaded zipped archive can be used during pipeline execution. If the archive includes a directory structure, the structure is preserved.

   > **IMPORTANT**
   > Please use unique and meaningful name for files in the script bundle since some common words (like `test`, `app` and etc) are reserved for built-in services.

   Following is a script bundle example, which contains a Python script file and a txt file:



   Following is the content of `my_script.py`:

```
def my_func(dataframe1):
    return dataframe1
```

   Following is sample code showing how to consume the files in the script bundle:

```
import pandas as pd
from my_script import my_func

def azureml_main(dataframe1 = None, dataframe2 = None):

    # Execution logic goes here
    print(f'Input pandas.DataFrame #1: {dataframe1}')

    # Test the custom defined Python function
    dataframe1 = my_func(dataframe1)

    # Test to read custom uploaded files by relative path
    with open('./Script Bundle/my_sample.txt', 'r') as text_file:
        sample = text_file.read()

    return dataframe1, pd.DataFrame(columns=["Sample"], data=[[sample]])
```

4. In the **Python script** text box, type or paste valid Python script.

> **NOTE**
> Be careful when writing your script. Make sure there are no syntax errors, such as using undeclared variables or unimported components or functions. Pay extra attention to the preinstalled component list. To import components that aren't listed, install the corresponding packages in your script, such as:
>
> ```
> import os
> os.system(f"pip install scikit-misc")
> ```

The **Python script** text box is prepopulated with some instructions in comments, and sample code for data access and output. You must edit or replace this code. Follow Python conventions for indentation and casing:

- The script must contain a function named `azureml_main` as the entry point for this component.
- The entry point function must have two input arguments, `Param<dataframe1>` and `Param<dataframe2>`, even when these arguments aren't used in your script.
- 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`.

If your .zip file contains `mymodule.py`, import it by using `import mymodule`.

Two datasets can be returned to the designer, 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.

> **WARNING**
> It's **Not** recommended to connect to a Database or other external storages in **Execute Python Script component**. You can use Import Data component and Export Data component

5. Submit the pipeline.

If the component is completed, check the output if as expected.

If the component is failed, you need to do some troubleshooting. Select the component, and open **Outputs+logs** in the right pane. Open **70_driver_log.txt** and search **in azureml_main**, then you could find which line caused the error. For example, "File "/tmp/tmp01_ID/user_script.py", line 17, in azureml_main" indicates that the error occurred in the 17 line of your Python script.

Results

The results of any computations by the embedded Python code must be provided as `pandas.DataFrame`, which is automatically converted to the Azure Machine Learning dataset format. You can then use the results with other components in the pipeline.

The component returns two datasets:

- **Results Dataset 1**, defined by the first returned pandas data frame in a Python script.
- **Result Dataset 2**, defined by the second returned pandas data frame in a Python script.

Preinstalled Python packages

The preinstalled packages are:

- adal==1.2.2
- applicationinsights==0.11.9
- attrs==19.3.0
- azure-common==1.1.25
- azure-core==1.3.0
- azure-graphrbac==0.61.1
- azure-identity==1.3.0
- azure-mgmt-authorization==0.60.0
- azure-mgmt-containerregistry==2.8.0
- azure-mgmt-keyvault==2.2.0
- azure-mgmt-resource==8.0.1
- azure-mgmt-storage==8.0.0
- azure-storage-blob==1.5.0
- azure-storage-common==1.4.2
- azureml-core==1.1.5.5
- azureml-dataprep-native==14.1.0
- azureml-dataprep==1.3.5
- azureml-defaults==1.1.5.1
- azureml-designer-classic-modules==0.0.118
- azureml-designer-core==0.0.31
- azureml-designer-internal==0.0.18
- azureml-model-management-sdk==1.0.1b6.post1
- azureml-pipeline-core==1.1.5
- azureml-telemetry==1.1.5.3
- backports.tempfile==1.0
- backports.weakref==1.0.post1
- boto3==1.12.29
- botocore==1.15.29
- cachetools==4.0.0
- certifi==2019.11.28
- cffi==1.12.3
- chardet==3.0.4
- click==7.1.1
- cloudpickle==1.3.0

- configparser==3.7.4
- contextlib2==0.6.0.post1
- cryptography==2.8
- cycler==0.10.0
- dill==0.3.1.1
- distro==1.4.0
- docker==4.2.0
- docutils==0.15.2
- dotnetcore2==2.1.13
- flask==1.0.3
- fusepy==3.0.1
- gensim==3.8.1
- google-api-core==1.16.0
- google-auth==1.12.0
- google-cloud-core==1.3.0
- google-cloud-storage==1.26.0
- google-resumable-media==0.5.0
- googleapis-common-protos==1.51.0
- gunicorn==19.9.0
- idna==2.9
- imbalanced-learn==0.4.3
- isodate==0.6.0
- itsdangerous==1.1.0
- jeepney==0.4.3
- jinja2==2.11.1
- jmespath==0.9.5
- joblib==0.14.0
- json-logging-py==0.2
- jsonpickle==1.3
- jsonschema==3.0.1
- kiwisolver==1.1.0
- liac-arff==2.4.0
- lightgbm==2.2.3
- markupsafe==1.1.1
- matplotlib==3.1.3
- more-itertools==6.0.0
- msal-extensions==0.1.3
- msal==1.1.0
- msrest==0.6.11
- msrestazure==0.6.3
- ndg-httpsclient==0.5.1
- nimbusml==1.6.1
- numpy==1.18.2
- oauthlib==3.1.0
- pandas==0.25.3
- pathspec==0.7.0

- pip==20.0.2
- portalocker==1.6.0
- protobuf==3.11.3
- pyarrow==0.16.0
- pyasn1-modules==0.2.8
- pyasn1==0.4.8
- pycparser==2.20
- pycryptodomex==3.7.3
- pyjwt==1.7.1
- pyopenssl==19.1.0
- pyparsing==2.4.6
- pyrsistent==0.16.0
- python-dateutil==2.8.1
- pytz==2019.3
- requests-oauthlib==1.3.0
- requests==2.23.0
- rsa==4.0
- ruamel.yaml==0.15.89
- s3transfer==0.3.3
- scikit-learn==0.22.2
- scipy==1.4.1
- secretstorage==3.1.2
- setuptools==46.1.1.post20200323
- six==1.14.0
- smart-open==1.10.0
- urllib3==1.25.8
- websocket-client==0.57.0
- werkzeug==0.16.1
- wheel==0.34.2

Next steps

See the set of components available to Azure Machine Learning.

Execute R Script component

3/28/2022 • 11 minutes to read • Edit Online

This article describes how to use the Execute R Script component to run R code in your Azure Machine Learning designer pipeline.

With R, you can do tasks that aren't supported by existing components, such as:

- Create custom data transformations
- Use your own metrics to evaluate predictions
- Build models using algorithms that aren't implemented as standalone components in the designer

R version support

Azure Machine Learning designer uses the CRAN (Comprehensive R Archive Network) distribution of R. The currently used version is CRAN 3.5.1.

Supported R packages

The R environment is preinstalled with more than 100 packages. For a complete list, see the section Preinstalled R packages.

You can also add the following code to any Execute R Script component, to see the installed packages.

```
azureml_main <- function(dataframe1, dataframe2){
  print("R script run.")
  dataframe1 <- data.frame(installed.packages())
  return(list(dataset1=dataframe1, dataset2=dataframe2))
}
```

NOTE

If your pipeline contains multiple Execute R Script components that need packages that aren't in the preinstalled list, install the packages in each component.

Installing R packages

To install additional R packages, use the `install.packages()` method. Packages are installed for each Execute R Script component. They aren't shared across other Execute R Script components.

NOTE

It's **NOT** recommended to install R package from the script bundle. It's recommended to install packages directly in the script editor. Specify the CRAN repository when you're installing packages, such as `install.packages("zoo",repos = "https://cloud.r-project.org")`.

WARNING

Excute R Script component does not support installing packages that require native compilation, like `qdap` package which requires JAVA and `drc` package which requires C++. This is because this component is executed in a pre-installed

environment with non-admin permission. Do not install packages which are pre-built on/for Windows, since the designer components are running on Ubuntu. To check whether a package is pre-built on windows, you could go to CRAN and search your package, download one binary file according to your OS, and check **Built:** part in the **DESCRIPTION** file. Following is an example:

abind: Combine Multidimensional Arrays

Combine multidimensional arrays into a single array. This is a generalization of 'cbind' ar
'adrop', 'asub', and 'afill' for manipulating, extracting and replacing data in arrays.

Version:	1.4-5
Depends:	R (≥ 1.5.0)
Imports:	methods, utils
Published:	2016-07-21
Author:	Tony Plate and Richard Heiberger
Maintainer:	Tony Plate <tplate at acm.org>
License:	LGPL-2 | LGPL-2.1 | LGPL-3 [expanded from: LGPL (≥ 2)]
NeedsCompilation:	no
Materials:	ChangeLog
In views:	Multivariate
CRAN checks:	abind results

更多功能

Downloads:

Reference manual:	abind.pdf
Package source:	abind_1.4-5.tar.gz
Windows binaries:	r-devel: abind_1.4-5.zip, r-release: abind_1.4-5.zip, r-oldrel: abind_1.
macOS binaries:	r-release: abind_1.4-5.tgz, r-oldrel: abind_1.4-5.tgz
Old sources:	abind archive

Name	Type	Compressed size	Password p...	Size
help	File folder			
html	File folder			
Meta	File folder			
R	File folder			
DESCRIPTION	File	1 KB	No	
INDEX	File	1 KB	No	

DESCRIPTION - Notepad

File Edit Format View Help

```
Package: abind
Version: 1.4-5
Date: 2016-06-19
Title: Combine Multidimensional Arrays
Author: Tony Plate <tplate@acm.org> and Richard Heiberger
Maintainer: Tony Plate <tplate@acm.org>
Description: Combine multidimensional arrays into a single array.
  This is a generalization of 'cbind' and 'rbind'.  Works with
  vectors, matrices, and higher-dimensional arrays.  Also
  provides functions 'adrop', 'asub', and 'afill' for manipulating,
  extracting and replacing data in arrays.
Depends: R (>= 1.5.0)
Imports: methods, utils
License: LGPL (>= 2)
NeedsCompilation: no
Packaged: 2016-07-19 15:24:25 UTC; tap
Repository: CRAN
Date/Publication: 2016-07-21 19:18:05
Built: R 4.0.3; ; 2020-10-15 18:10:45 UTC; windows
```

This sample shows how to install Zoo:

```
# R version: 3.5.1
# The script MUST contain a function named azureml_main,
# which is the entry point for this component.

# Note that functions dependent on the X11 library,
# such as "View," are not supported because the X11 library
# is not preinstalled.

# The entry point function MUST have two input arguments.
# If the input port is not connected, the corresponding
# dataframe argument will be null.
#   Param<dataframe1>: a R DataFrame
#   Param<dataframe2>: a R DataFrame
azureml_main <- function(dataframe1, dataframe2){
  print("R script run.")

  if(!require(zoo)) install.packages("zoo",repos = "https://cloud.r-project.org")
  library(zoo)
  # Return datasets as a Named List
  return(list(dataset1=dataframe1, dataset2=dataframe2))
}
```

NOTE

Before you install a package, check if it already exists so you don't repeat an installation. Repeat installations might cause web service requests to time out.

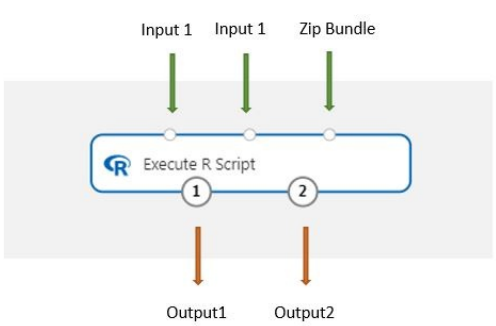
Access to registered dataset

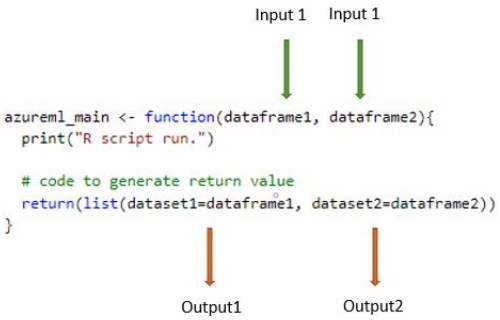
You can refer to the following sample code to access to the registered datasets in your workspace:

```
azureml_main <- function(dataframe1, dataframe2){
  print("R script run.")
  run = get_current_run()
  ws = run$experiment$workspace
  dataset = azureml$core$dataset$Dataset$get_by_name(ws, "YOUR DATASET NAME")
  dataframe2 <- dataset$to_pandas_dataframe()
  # Return datasets as a Named List
  return(list(dataset1=dataframe1, dataset2=dataframe2))
}
```

How to configure Execute R Script

The Execute R Script component contains sample code as a starting point.





Datasets stored in the designer are automatically converted to an R data frame when loaded with this component.

1. Add the **Execute R Script** component to your pipeline.
2. Connect any inputs that the script needs. Inputs are optional and can include data and additional R code.
   - **Dataset1**: Reference the first input as `dataframe1`. The input dataset must be formatted as a CSV, TSV, or ARFF file. Or you can connect an Azure Machine Learning dataset.
   - **Dataset2**: Reference the second input as `dataframe2`. This dataset also must be formatted as a CSV, TSV, or ARFF file, or as an Azure Machine Learning dataset.
   - **Script Bundle**: The third input accepts .zip files. A zipped file can contain multiple files and multiple file types.
3. In the **R script** text box, type or paste valid R script.

   > **NOTE**
   > Be careful when writing your script. Make sure there are no syntax errors, such as using undeclared variables or unimported components or functions. Pay extra attention to the preinstalled package list at the end of this article. To use packages that aren't listed, install them in your script. An example is `install.packages("zoo",repos = "https://cloud.r-project.org")`.

   To help you get started, the **R Script** text box is prepopulated with sample code, which you can edit or replace.

```
# R version: 3.5.1
# The script MUST contain a function named azureml_main,
# which is the entry point for this component.

# Note that functions dependent on the X11 library,
# such as "View," are not supported because the X11 library
# is not preinstalled.

# The entry point function MUST have two input arguments.
# If the input port is not connected, the corresponding
# dataframe argument will be null.
#   Param<dataframe1>: a R DataFrame
#   Param<dataframe2>: a R DataFrame
azureml_main <- function(dataframe1, dataframe2){
print("R script run.")

# If a .zip file is connected to the third input port, it's
# unzipped under "./Script Bundle". This directory is added
# to sys.path.

# Return datasets as a Named List
return(list(dataset1=dataframe1, dataset2=dataframe2))
}
```

The entry point function must have the input arguments `Param<dataframe1>` and `Param<dataframe2>`, even when these arguments aren't used in the function.

> **NOTE**
> The data passed to the Execute R Script component is referenced as `dataframe1` and `dataframe2`, which is different from Azure Machine Learning designer (the designer reference as `dataset1`, `dataset2`). Make sure that input data is referenced correctly in your script.

> **NOTE**
> Existing R code might need minor changes to run in a designer pipeline. 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 the designer.

4. If your script is larger than 16 KB, use the **Script Bundle** port to avoid errors like *CommandLine exceeds the limit of 16597 characters*.

   a. Bundle the script and other custom resources to a zip file.
   b. Upload the zip file as a **File Dataset** to the studio.
   c. Drag the dataset component from the *Datasets* list in the left component pane in the designer authoring page.
   d. Connect the dataset component to the **Script Bundle** port of **Execute R Script** component.

   Following is the sample code to consume the script in the script bundle:

```
azureml_main <- function(dataframe1, dataframe2){
# Source the custom R script: my_script.R
source("./Script Bundle/my_script.R")

# Use the function that defined in my_script.R
dataframe1 <- my_func(dataframe1)

sample <- readLines("./Script Bundle/my_sample.txt")
return (list(dataset1=dataframe1, dataset2=data.frame("Sample"=sample)))
}
```

5. For **Random Seed**, enter 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.

6. Submit the pipeline.

Results

Execute R Script components can return multiple outputs, but they must be provided as R data frames. The designer automatically converts data frames to datasets for compatibility with other components.

Standard messages and errors from R are returned to the component's log.

If you need to print results in the R script, you can find the printed results in **70_driver_log** under the **Outputs+logs** tab in the right panel of the component.

Sample scripts

There are many ways to extend your pipeline by using custom R scripts. This section provides sample code for common tasks.

Add an R script as an input

The Execute R Script component supports arbitrary R script files as inputs. To use them, you must upload them to your workspace as part of the .zip file.

1. To upload a .zip file that contains R code to your workspace, go to the **Datasets** asset page. Select **Create dataset**, and then select **From local file** and the **File** dataset type option.

2. Verify that the zipped file appears in **My Datasets** under the **Datasets** category in the left component tree.

3. Connect the dataset to the **Script Bundle** input port.

4. All files in the .zip file are available during pipeline run time.

   If the script bundle file contained a directory structure, the structure is preserved. But you must alter your code to prepend the directory **./Script Bundle** to the path.

Process data

The following sample shows how to scale and normalize input data:

```
# R version: 3.5.1
# The script MUST contain a function named azureml_main,
# which is the entry point for this component.

# Note that functions dependent on the X11 library,
# such as "View," are not supported because the X11 library
# is not preinstalled.

# The entry point function MUST have two input arguments.
# If the input port is not connected, the corresponding
# dataframe argument will be null.
#   Param<dataframe1>: a R DataFrame
#   Param<dataframe2>: a R DataFrame
azureml_main <- function(dataframe1, dataframe2){
  print("R script run.")
  # If a .zip file is connected to the third input port, it's
  # unzipped under "./Script Bundle". This directory is added
  # to sys.path.
  series <- dataframe1$width
  # Find the maximum and minimum values of the width column in dataframe1
  max_v <- max(series)
  min_v <- min(series)
  # Calculate the scale and bias
  scale <- max_v - min_v
  bias <- min_v / dis
  # Apply min-max normalizing
  dataframe1$width <- dataframe1$width / scale - bias
  dataframe2$width <- dataframe2$width / scale - bias
  # Return datasets as a Named List
  return(list(dataset1=dataframe1, dataset2=dataframe2))
}
```

Read a .zip file as input

This sample shows how to use a dataset in a .zip file as an input to the Execute R Script component.

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.
4. Connect the resulting dataset to the **Script Bundle** input of your **Execute R Script** component.
5. Use the following code to read the CSV data from the zipped file.

```
azureml_main <- function(dataframe1, dataframe2){
  print("R script run.")
  mydataset<-read.csv("./Script Bundle/mydatafile.csv",encoding="UTF-8");
  # Return datasets as a Named List
  return(list(dataset1=mydataset, dataset2=dataframe2))
}
```

Replicate rows

This sample shows how to replicate positive records in a dataset to balance the sample:

```
azureml_main <- function(dataframe1, dataframe2){
  data.set <- dataframe1[dataframe1[,1]==-1,]
  # positions of the positive samples
  pos <- dataframe1[dataframe1[,1]==1,]
  # replicate the positive samples to balance the sample
  for (i in 1:20) data.set <- rbind(data.set,pos)
  row.names(data.set) <- NULL
  # Return datasets as a Named List
  return(list(dataset1=data.set, dataset2=dataframe2))
}
```

Pass R objects between Execute R Script components

You can pass R objects between instances of the Execute R Script component by using the internal serialization mechanism. This example assumes that you want to move the R object named `A` between two Execute R Script components.

1. Add the first **Execute R Script** component to your pipeline. Then enter the following code in the **R Script** text box to create a serialized object `A` as a column in the component's output data table:

   ```
   azureml_main <- function(dataframe1, dataframe2){
     print("R script run.")
     # some codes generated A

     serialized <- as.integer(serialize(A,NULL))
     data.set <- data.frame(serialized,stringsAsFactors=FALSE)

     return(list(dataset1=data.set, dataset2=dataframe2))
   }
   ```

   The explicit conversion to integer type is done because the serialization function outputs data in the R `Raw` format, which the designer doesn't support.

2. Add a second instance of the **Execute R Script** component, and connect it to the output port of the previous component.

3. Type the following code in the **R Script** text box to extract object `A` from the input data table.

   ```
   azureml_main <- function(dataframe1, dataframe2){
     print("R script run.")
     A <- unserialize(as.raw(dataframe1$serialized))
     # Return datasets as a Named List
     return(list(dataset1=dataframe1, dataset2=dataframe2))
   }
   ```

Preinstalled R packages

The following preinstalled R packages are currently available:

PACKAGE	VERSION
askpass	1.1
assertthat	0.2.1
backports	1.1.4

PACKAGE	VERSION
base	3.5.1
base64enc	0.1-3
BH	1.69.0-1
bindr	0.1.1
bindrcpp	0.2.2
bitops	1.0-6
boot	1.3-22
broom	0.5.2
callr	3.2.0
caret	6.0-84
caTools	1.17.1.2
cellranger	1.1.0
class	7.3-15
cli	1.1.0
clipr	0.6.0
cluster	2.0.7-1
codetools	0.2-16
colorspace	1.4-1
compiler	3.5.1
crayon	1.3.4
curl	3.3
data.table	1.12.2
datasets	3.5.1
DBI	1.0.0
dbplyr	1.4.1

PACKAGE	VERSION
digest	0.6.19
dplyr	0.7.6
e1071	1.7-2
evaluate	0.14
fansi	0.4.0
forcats	0.3.0
foreach	1.4.4
foreign	0.8-71
fs	1.3.1
gdata	2.18.0
generics	0.0.2
ggplot2	3.2.0
glmnet	2.0-18
glue	1.3.1
gower	0.2.1
gplots	3.0.1.1
graphics	3.5.1
grDevices	3.5.1
grid	3.5.1
gtable	0.3.0
gtools	3.8.1
haven	2.1.0
highr	0.8
hms	0.4.2
htmltools	0.3.6

PACKAGE	VERSION
httr	1.4.0
ipred	0.9-9
iterators	1.0.10
jsonlite	1.6
KernSmooth	2.23-15
knitr	1.23
labeling	0.3
lattice	0.20-38
lava	1.6.5
lazyeval	0.2.2
lubridate	1.7.4
magrittr	1.5
markdown	1
MASS	7.3-51.4
Matrix	1.2-17
methods	3.5.1
mgcv	1.8-28
mime	0.7
ModelMetrics	1.2.2
modelr	0.1.4
munsell	0.5.0
nlme	3.1-140
nnet	7.3-12
numDeriv	2016.8-1.1
openssl	1.4

PACKAGE	VERSION
parallel	3.5.1
pillar	1.4.1
pkgconfig	2.0.2
plogr	0.2.0
plyr	1.8.4
prettyunits	1.0.2
processx	3.3.1
prodlim	2018.04.18
progress	1.2.2
ps	1.3.0
purrr	0.3.2
quadprog	1.5-7
quantmod	0.4-15
R6	2.4.0
randomForest	4.6-14
RColorBrewer	1.1-2
Rcpp	1.0.1
RcppRoll	0.3.0
readr	1.3.1
readxl	1.3.1
recipes	0.1.5
rematch	1.0.1
reprex	0.3.0
reshape2	1.4.3
reticulate	1.12

PACKAGE	VERSION
rlang	0.4.0
rmarkdown	1.13
ROCR	1.0-7
rpart	4.1-15
rstudioapi	0.1
rvest	0.3.4
scales	1.0.0
selectr	0.4-1
spatial	7.3-11
splines	3.5.1
SQUAREM	2017.10-1
stats	3.5.1
stats4	3.5.1
stringi	1.4.3
stringr	1.3.1
survival	2.44-1.1
sys	3.2
tcltk	3.5.1
tibble	2.1.3
tidyr	0.8.3
tidyselect	0.2.5
tidyverse	1.2.1
timeDate	3043.102
tinytex	0.13
tools	3.5.1

PACKAGE	VERSION
tseries	0.10-47
TTR	0.23-4
utf8	1.1.4
utils	3.5.1
vctrs	0.1.0
viridisLite	0.3.0
whisker	0.3-2
withr	2.1.2
xfun	0.8
xml2	1.2.0
xts	0.11-2
yaml	2.2.0
zeallot	0.1.0
zoo	1.8-6

Next steps

See the set of components available to Azure Machine Learning.

Convert Word to Vector component

3/28/2022 • 6 minutes to read • Edit Online

This article describes how to use the Convert Word to Vector component in Azure Machine Learning designer to do these tasks:

- Apply various Word2Vec models (Word2Vec, FastText, GloVe pretrained model) on the corpus of text that you specified as input.
- Generate a vocabulary with word embeddings.

This component uses the Gensim library. For more information about Gensim, see its official website, which includes tutorials and an explanation of algorithms.

More about converting words to vectors

Converting words to vectors, or word vectorization, is a natural language processing (NLP) process. The process uses language models to map words into vector space. A vector space represents each word by a vector of real numbers. It also allows words with similar meanings have similar representations.

Use word embeddings as initial input for NLP downstream tasks such as text classification and sentiment analysis.

Among various word embedding technologies, in this component, we implemented three widely used methods. Two, Word2Vec and FastText, are online-training models. The other is a pretrained model, glove-wiki-gigaword-100.

Online-training models are trained on your input data. Pretrained models are trained offline on a larger text corpus (for example, Wikipedia, Google News) that usually contains about 100 billion words. Word embedding then stays constant during word vectorization. Pretrained word models provide benefits such as reduced training time, better word vectors encoded, and improved overall performance.

Here's some information about the methods:

- Word2Vec is one of the most popular techniques to learn word embeddings by using a shallow neural network. The theory is discussed in this paper, available as a PDF download: Efficient Estimation of Word Representations in Vector Space. The implementation in this component is based on the Gensim library for Word2Vec.
- The FastText theory is explained in this paper, available as a PDF download: Enriching Word Vectors with Subword Information. The implementation in this component is based on the Gensim library for FastText.
- The GloVe pretrained model is glove-wiki-gigaword-100. It's a collection of pretrained vectors based on a Wikipedia text corpus, which contains 5.6 billion tokens and 400,000 uncased vocabulary words. A PDF download is available: GloVe: Global Vectors for Word Representation.

How to configure Convert Word to Vector

This component requires a dataset that contains a column of text. Preprocessed text is better.

1. Add the **Convert Word to Vector** component to your pipeline.
2. As input for the component, provide a dataset that contains one or more text columns.
3. For **Target column**, choose only one column that contains text to process.

Because this component creates a vocabulary from text, the content of columns differs, which leads to different vocabulary contents. That's why the component accepts only one target column.

4. For **Word2Vec strategy**, choose from **GloVe pretrained English Model**, **Gensim Word2Vec**, and **Gensim FastText**.

5. If **Word2Vec strategy** is **Gensim Word2Vec** or **Gensim FastText**:

   - For **Word2Vec Training Algorithm**, choose from **Skip_gram** and **CBOW**. The difference is introduced in the original paper (PDF).

     The default method is **Skip_gram**.

   - For **Length of word embedding**, specify the dimensionality of the word vectors. This setting corresponds to the `size` parameter in Gensim.

     The default embedding size is 100.

   - For **Context window size**, specify the maximum distance between the word being predicted and the current word. This setting corresponds to the `window` parameter in Gensim.

     The default window size is 5.

   - For **Number of epochs**, specify the number of epochs (iterations) over the corpus. Corresponds to the `iter` parameter in Gensim.

     The default epoch number is 5.

6. For **Maximum vocabulary size**, specify the maximum number of the words in the generated vocabulary.

   If there are more unique words than the max size, prune the infrequent ones.

   The default vocabulary size is 10,000.

7. For **Minimum word count**, provide a minimum word count. The component will ignore all words that have a frequency lower than this value.

   The default value is 5.

8. Submit the pipeline.

Examples

The component has one output:

- **Vocabulary with embeddings**: Contains the generated vocabulary, together with each word's embedding. One dimension occupies one column.

The following example shows how the Convert Word to Vector component works. It uses Convert Word to Vector with default settings to the preprocessed Wikipedia SP 500 Dataset.

Source dataset

The dataset contains a category column, along with the full text fetched from Wikipedia. The following table shows a few representative examples.

TEXT
nasdaq 100 component s p 500 component foundation founder location city apple campus 1 infinite loop street infinite loop cupertino california cupertino california location country united states...

TEXT
br nasdaq 100 nasdaq 100 component br s p 500 s p 500 component industry computer software foundation br founder charles geschke br john warnock location adobe systems...
s p 500 s p 500 component industry automotive industry automotive predecessor general motors corporation 1908 2009 successor...
s p 500 s p 500 component industry conglomerate company conglomerate foundation founder location city fairfield connecticut fairfield connecticut location country usa area...
br s p 500 s p 500 component foundation 1903 founder william s harley br arthur davidson harley davidson founder arthur davidson br walter davidson br william a davidson location...

Output vocabulary with embeddings

The following table contains the output of this component, taking the Wikipedia SP 500 dataset as input. The leftmost column indicates the vocabulary. Its embedding vector is represented by values of remaining columns in the same row.

VOCABULARY	EMBEDDING DIM 0	EMBEDDING DIM 1	EMBEDDING DIM 2	EMBEDDING DIM 3	EMBEDDING DIM 4	EMBEDDING DIM 5	...	EMBEDDING DIM 99
nasdaq	-0.375865	0.609234	0.812797	-0.002236	0.319071	-0.591986	...	0.364276
component	0.081302	0.40001	0.121803	0.108181	0.043651	-0.091452	...	0.636587
s	-0.34355	-0.037092	-0.012167	0.151542	0.601019	0.084501	...	0.149419
p	-0.133407	0.073244	0.170396	0.326706	0.213463	-0.700355	...	0.530901
foundation	-0.166819	0.10883	-0.07933	-0.073753	0.262137	0.045725	...	0.27487
founder	-0.297408	0.493067	0.316709	-0.031651	0.455416	-0.284208	...	0.22798
location	-0.375213	0.461229	0.310698	0.213465	0.200092	0.314288	...	0.14228
city	-0.460828	0.505516	-0.074294	-0.00639	0.116545	0.494368	...	-0.2403
apple	0.05779	0.672657	0.597267	-0.898889	0.099901	0.11833	...	0.4636
campus	-0.281835	0.29312	0.106966	-0.031385	0.100777	-0.061452	...	0.05978
infinite	-0.263074	0.245753	0.07058	-0.164666	0.162857	-0.027345	...	-0.0525

VOCABULARY	EMBEDDING DIM 0	EMBEDDING DIM 1	EMBEDDING DIM 2	EMBEDDING DIM 3	EMBEDDING DIM 4	EMBEDDING DIM 5	...	EMBEDDING DIM 99
loop	-0.391421	0.52366	0.141503	-0.105423	0.084503	-0.018424	...	-0.0521

In this example, we used the default **Gensim Word2Vec** for **Word2Vec strategy**, and **Training Algorithm** is **Skip-gram**. **Length of word Embedding** is 100, so we have 100 embedding columns.

Technical notes

This section contains tips and answers to frequently asked questions.

- Difference between online-training and pretrained model:

  In this Convert Word to Vector component, we provided three different strategies: two online-training models and one pretrained model. The online-training models use your input dataset as training data, and generate vocabulary and word vectors during training. The pretrained model is already trained by a much larger text corpus, such as Wikipedia or Twitter text. The pretrained model is actually a collection of word/embedding pairs.

  The GloVe pre-trained model summarizes a vocabulary from the input dataset and generates an embedding vector for each word from the pretrained model. Without online training, the use of a pretrained model can save training time. It has better performance, especially when the input dataset size is relatively small.

- Embedding size:

  In general, the length of word embedding is set to a few hundred. For example, 100, 200, 300. A small embedding size means a small vector space, which could cause word embedding collisions.

  The length of word embeddings is fixed for pretrained models. In this example, the embedding size of glove-wiki-gigaword-100 is 100.

Next steps

See the set of components available to Azure Machine Learning.

For a list of errors specific to the designer components, see Machine Learning error codes.

Extract N-Gram Features from Text component reference

3/28/2022 • 6 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer. Use the Extract N-Gram Features from Text component to *featurize* unstructured text data.

Configuration of the Extract N-Gram Features from Text component

The component supports the following scenarios for using an n-gram dictionary:

- Create a new n-gram dictionary from a column of free text.
- Use an existing set of text features to featurize a free text column.
- Score or deploy a model that uses n-grams.

Create a new n-gram dictionary

1. Add the Extract N-Gram Features from Text component to your pipeline, and connect the dataset that has the text you want to process.
2. Use **Text column** to choose a column of string type that contains the text you want to extract. Because results are verbose, you can process only a single column at a time.
3. Set **Vocabulary mode** to **Create** to indicate that you're creating a new list of n-gram features.
4. Set **N-Grams size** to indicate the *maximum* size of the n-grams to extract and store.

   For example, if you enter 3, unigrams, bigrams, and trigrams will be created.
5. **Weighting function** specifies how to build the document feature vector and how to extract vocabulary from documents.
   - **Binary Weight**: Assigns a binary presence value to the extracted n-grams. The value for each n-gram is 1 when it exists in the document, and 0 otherwise.
   - **TF Weight**: Assigns a term frequency (TF) score to the extracted n-grams. The value for each n-gram is its occurrence frequency in the document.
   - **IDF Weight**: Assigns an inverse document frequency (IDF) score 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.

     ```
     IDF = log of corpus_size / document_frequency
     ```
   - **TF-IDF Weight**: Assigns a term frequency/inverse document frequency (TF/IDF) score to the extracted n-grams. The value for each n-gram is its TF score multiplied by its IDF score.
6. Set **Minimum word length** to the minimum number of letters that can be used in any *single word* in an n-gram.
7. Use **Maximum word length** to set 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.

8. Use **Minimum n-gram document absolute frequency** to set the minimum occurrences required for any n-gram to be included in the n-gram dictionary.

   For example, if you use the default value of 5, any n-gram must appear at least five times in the corpus to be included in the n-gram dictionary.

9. Set **Maximum n-gram document ratio** to the maximum ratio of 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. It varies from document to document. For example, if you're 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.

10. Select the option **Normalize n-gram feature vectors** to normalize the feature vectors. If this option is enabled, each n-gram feature vector is divided by its L2 norm.

11. Submit the pipeline.

Use an existing n-gram dictionary

1. Add the Extract N-Gram Features from Text component to your pipeline, and connect the dataset that has the text you want to process to the **Dataset** port.

2. Use **Text column** to select the text column that contains the text you want to featurize. By default, the component selects all columns of type **string**. For best results, process a single column at a time.

3. Add the saved dataset that contains 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 component.

4. For **Vocabulary mode**, select the **ReadOnly** update option from the drop-down list.

   The **ReadOnly** option represents the input corpus for the input vocabulary. 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.

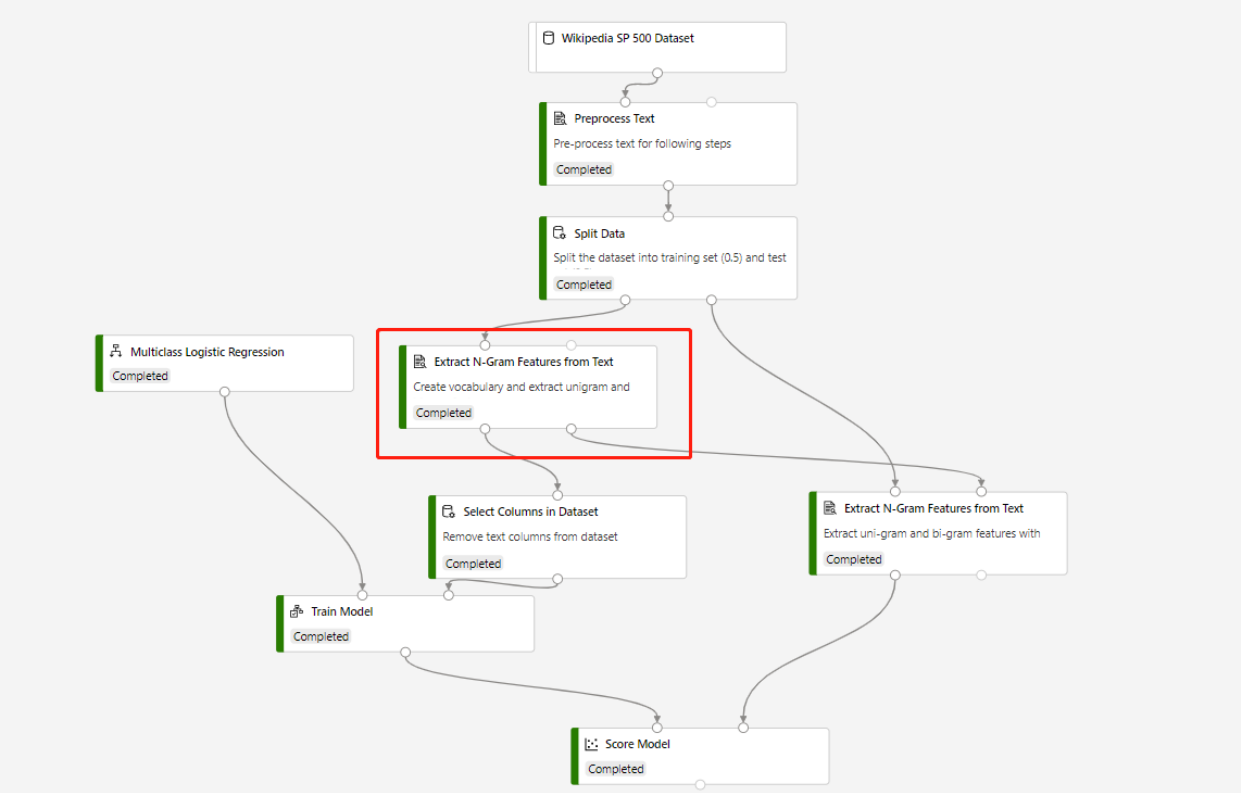
   > **TIP**
   > Use this option when you're scoring a text classifier.

5. For all other options, see the property descriptions in the previous section.

6. Submit the pipeline.

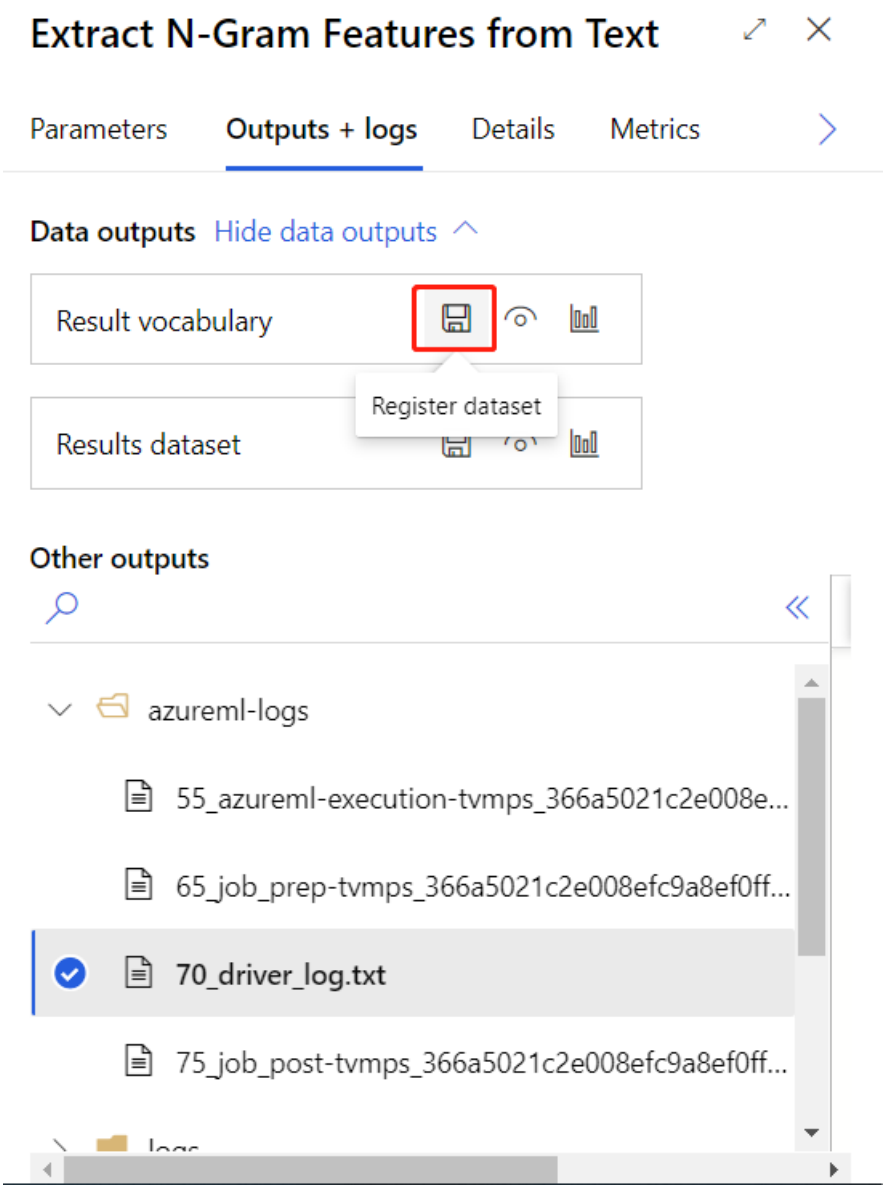
Build inference pipeline that uses n-grams to deploy a real-time endpoint

A training pipeline which contains **Extract N-Grams Feature From Text** and **Score Model** to make prediction on test dataset, is built in following structure:

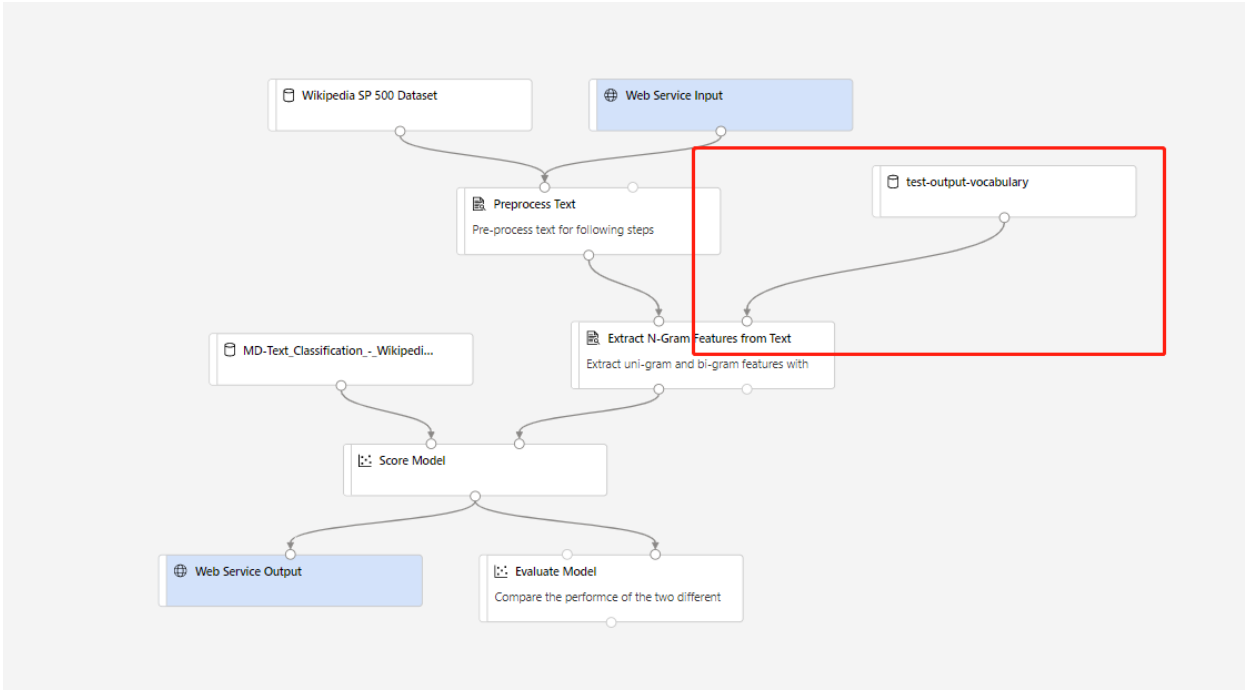


Vocabulary mode of the circled **Extract N-Grams Feature From Text** component is **Create**, and **Vocabulary mode** of the component which connects to **Score Model** component is **ReadOnly**.

After submitting the training pipeline above successfully, you can register the output of the circled component as dataset.



Then you can create real-time inference pipeline. After creating inference pipeline, you need to adjust your inference pipeline manually like following:



Then submit the inference pipeline, and deploy a real-time endpoint.

Results

The Extract N-Gram Features from Text component creates two types of output:

- **Result dataset**: This output is a summary of the analyzed text combined with the n-grams that were extracted. Columns that you didn't select in the **Text column** option are passed through to the output. For each column of text that you analyze, the component generates these columns:
  - **Matrix of n-gram occurrences**: The component 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 reuse with a different set of inputs, or for a later update. You can also reuse the vocabulary for modeling and scoring.

Result vocabulary

The vocabulary contains the n-gram dictionary with the term frequency scores that are generated as part of the analysis. The DF and IDF scores are generated regardless of other options.

- **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.

You can manually update this dataset, but you might introduce errors. For example:

- An error is raised if the component 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** column must be of the integer type.
- The **IDF** column must be of the float type.

> **NOTE**
> Don't connect the data output to the Train Model component directly. You should remove free text columns before they're fed into the Train Model. Otherwise, the free text columns will be treated as categorical features.

Next steps

See the set of components available to Azure Machine Learning.

Feature Hashing component reference

3/28/2022 • 4 minutes to read • Edit Online

This article describes a component included in Azure Machine Learning designer.

Use the Feature Hashing component to transform a stream of English text into a set of integer features. You can then pass this hashed feature set to a machine learning algorithm to train a text analytics model.

The feature hashing functionality provided in this component is based on the nimbusml framework. For more information, see NgramHash class.

What is 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.

USER TEXT	SENTIMENT
I loved this book	3
I hated this book	1
This book was great	3
I love books	2

Internally, the Feature Hashing component 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
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. The dictionary would also include single terms like these:

TERM (UNIGRAMS)	FREQUENCY
book	3
I	3

TERM (UNIGRAMS)	FREQUENCY
books	1
was	1

After the dictionary is built, the Feature Hashing component converts the dictionary terms into hash values. It then computes whether a feature was used in each case. For each row of text data, the component 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 didn't contain the hashed feature.
- If the value is 1, the row did contain the feature.

Feature hashing lets you represent text documents of variable length as numeric feature vectors of equal length to reduce dimensionality. If you tried to use the text column for training as is, it would be treated as a categorical feature column with many distinct values.

Numeric outputs also make it possible to use common machine learning methods, including classification, clustering, and information retrieval. Because lookup operations can use integer hashes rather than string comparisons, getting the feature weights is also much faster.

Configure the Feature Hashing component

1. Add the Feature Hashing component to your pipeline in the designer.
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 preprocessing text before you apply feature hashing.

3. Set **Target columns** to the text columns that you want to convert to hashed features. Keep in mind that:

   - The columns must be the string data type.
   - Choosing multiple text columns can have a significant impact on feature dimensionality. For example, the number of columns for a 10-bit hash goes from 1,024 for a single column to 2,048 for two columns.

4. Use **Hashing bitsize** to specify the number of bits to use when you're creating the hash table.

   The default bit size is 10. For many problems, this value is adequate. You might need more space to avoid collisions, depending on the size of the n-grams vocabulary in the training text.

5. For **N-grams**, enter 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.

For example, if you enter 3, unigrams, bigrams, and trigrams will be created.

6. Submit the pipeline.

Results

After processing is complete, the component outputs a transformed dataset in which the original text column has been converted to multiple columns. Each column represents a feature in the text. Depending on how significant the dictionary is, the resulting dataset can be 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 create the transformed dataset, you can use it as the input to the Train Model component.

Best practices

The following best practices can help you get the most out of the Feature Hashing component:

- Add a Preprocess Text component before using Feature Hashing to preprocess the input text.
- Add a Select Columns component after the Feature Hashing component to remove the text columns from the output dataset. You don't need the text columns after the hashing features have been generated.
- Consider using these text preprocessing options, to simplify results and improve accuracy:
    - Word breaking
    - Stopping word removal
    - Case normalization
    - Removal of punctuation and special characters
    - Stemming

The optimal set of preprocessing methods to apply in any solution depends on domain, vocabulary, and business need. pipeline with your data to see which text processing methods are most effective.

Next steps

See the set of components available to Azure Machine Learning

Preprocess Text

3/28/2022 • 3 minutes to read • Edit Online

This article describes a component in Azure Machine Learning designer.

Use the **Preprocess Text** component to clean and simplify text. It supports these common text processing operations:

- 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
- 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

The **Preprocess Text** component currently only supports English.

Configure Text Preprocessing

1. Add the **Preprocess Text** component to your pipeline in Azure Machine Learning. You can find this component under **Text Analytics**.
2. Connect a dataset that has at least one column containing text.
3. Select the language from the **Language** dropdown list.
4. **Text column to clean**: Select the column that you want to preprocess.
5. **Remove stop words**: Select this option if you want to apply a predefined stopword list to the text column.

   Stopword lists are language-dependent and customizable.
6. **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..
7. **Detect sentences**: Select this option if you want the component to insert a sentence boundary mark when performing analysis.

   This component uses a series of three pipe characters `|||` to represent the sentence terminator.
8. Perform optional find-and-replace operations using regular expressions. The regular expression will be processed at first, ahead of all other built-in options.
   - **Custom regular expression**: Define the text you're searching for.
   - **Custom replacement string**: Define a single replacement value.
9. **Normalize case to lowercase**: Select this option if you want to convert ASCII uppercase characters to their lowercase forms.

   If characters aren't normalized, the same word in uppercase and lowercase letters is considered two different words.

10. You can also 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 numbers are domain-dependent and language dependent. If numeric characters are an integral part of a known word, the number might not be removed. Learn more in Technical notes.
    - **Remove special characters**: Use this option to remove any non-alphanumeric special characters.
    - **Remove duplicate characters**: Select this option to remove extra characters in any sequences that repeat for more than twice. For example, a sequence like "aaaaa" would be reduced to "aa".
    - **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`

11. **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"*.

12. **Normalize backslashes to slashes**: Select this option to map all instances of `\\` to `/`.

13. **Split tokens on special characters**: Select this option if you want to break words on characters such as `&`, `-`, and so forth. This option can also reduce the special characters when it repeats more than twice.

    For example, the string `MS---WORD` would be separated into three tokens, `MS`, `-`, and `WORD`.

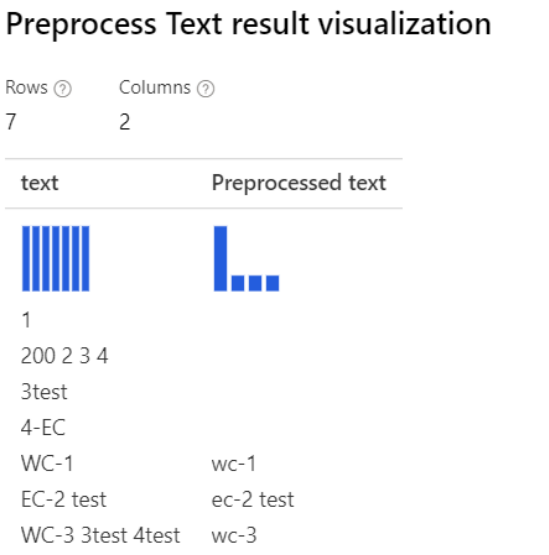
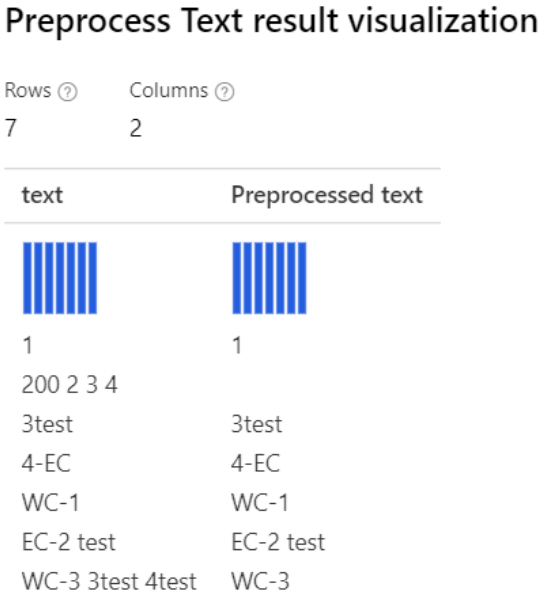
14. Submit the pipeline.

Technical notes

The **preprocess-text** component in Studio(classic) and designer use different language models. The designer uses a multi-task CNN trained model from spaCy. Different models give different tokenizer and part-of-speech tagger, which leads to different results.

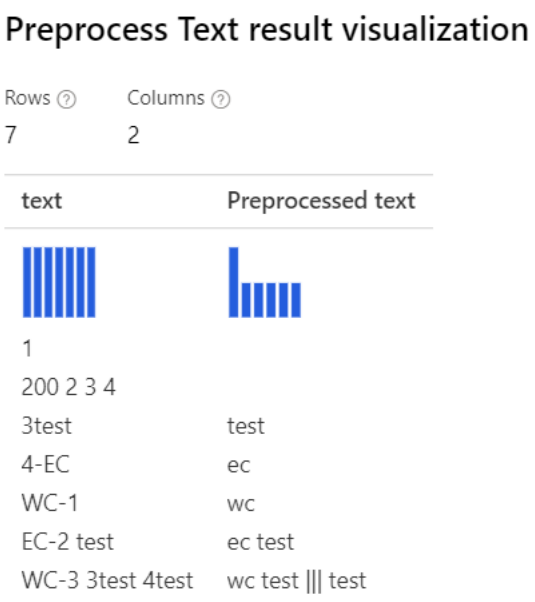
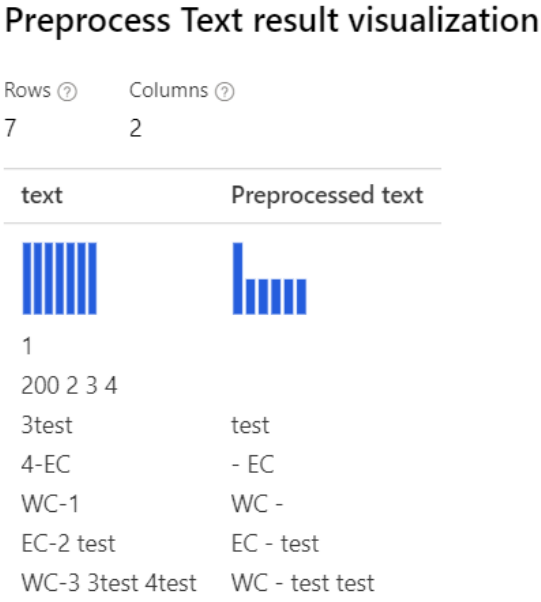
Following are some examples:

CONFIGURATION	OUTPUT RESULT

CONFIGURATION	OUTPUT RESULT
With all options selected Explanation: For the cases like '3test' in the 'WC-3 3test 4test', the designer remove the whole word '3test', since in this context, the part-of-speech tagger specifies this token '3test' as numeral, and according to the part-of-speech, the component removes it.	
With only `Removing number` selected Explanation: For the cases like '3test', '4-EC', the designer tokenizer dose not split these cases, and treats them as the whole tokens. So it won't remove the numbers in these words.	

You can also use regular expression to output customized results:

CONFIGURATION	OUTPUT RESULT

CONFIGURATION	OUTPUT RESULT
With all options selected Custom regular expression: `(\s+)*(-|\d+)(\s+)*` Custom replacement string: `\1 \2 \3`	
With only `Removing number` selected Custom regular expression: `(\s+)*(-|\d+)(\s+)*` Custom replacement string: `\1 \2 \3`	

Next steps

See the set of components available to Azure Machine Learning.

Latent Dirichlet Allocation component

3/28/2022 • 11 minutes to read • Edit Online

This article describes how to use the Latent Dirichlet Allocation component in Azure Machine Learning designer, to group otherwise unclassified text into categories.

Latent Dirichlet Allocation (LDA) is often used in natural language processing to find texts that are similar. Another common term is *topic modeling*.

This component takes a column of text and generates these outputs:

- The source text, together with a score for each category
- A feature matrix that contains extracted terms and coefficients for each category
- A transformation, which you can save and reapply to new text used as input

This component uses the scikit-learn library. For more information about scikit-learn, see the GitHub repository, which includes tutorials and an explanation of the algorithm.

More about Latent Dirichlet Allocation

LDA is generally not a method for classification. But it uses a generative approach, so you don't need to provide known class labels and then infer the patterns. Instead, the algorithm generates a probabilistic model that's 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.

You might prefer a generative model because it avoids making strong assumptions about the relationship between the text and categories. It 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 component is based on the scikit-learn library for LDA.

For more information, see the Technical notes section.

How to configure Latent Dirichlet Allocation

This component requires a dataset that contains a column of text, either raw or preprocessed.

1. Add the **Latent Dirichlet Allocation** component to your pipeline.
2. As input for the component, provide a dataset that contains one or more text columns.
3. For **Target columns**, choose one or more columns that contain text to analyze.

   You can choose multiple columns, but they must be of the **string** data type.

   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**, enter 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 convert output values to probabilities.

   Rather than representing the transformed values as integers, values in the output and feature dataset will be transformed as follows:

   - Values in the dataset will be represented as a probability where `P(topic|document)`.
   - Values in the feature topic matrix will be represented as a probability where `P(word|topic)`.

   > **NOTE**
   > In Azure Machine Learning designer, the scikit-learn library no longer supports unnormalized *doc_topic_distr* output from version 0.19. In this component, the **Normalize** parameter can only be applied to *feature Topic matrix* output. *Transformed dataset* output is always normalized.

7. Select the option **Show all options**, and then set it to **TRUE** if you want to set the following advanced parameters.

   These parameters are specific to the scikit-learn implementation of LDA. There are some good tutorials about LDA in scikit-learn, as well as the official scikit-learn document.

   - **Rho parameter**. Provide a prior probability for the sparsity of topic distributions. This parameter corresponds to sklearn's `topic_word_prior` parameter. Use the value **1** if you expect that the distribution of words is flat; that is, all words are assumed equiprobable. If you think most words appear sparsely, you might set it to a lower value.
   - **Alpha parameter**. Specify a prior probability for the sparsity of per-document topic weights. This parameter corresponds to sklearn's `doc_topic_prior` parameter.
   - **Estimated number of documents**. Enter a number that represents your best estimate of the number of documents (rows) that will be processed. This parameter lets the component allocate a hash table of sufficient size. It corresponds to the `total_samples` parameter in scikit-learn.
   - **Size of the batch**. Enter a number that indicates how many rows to include in each batch of text sent to the LDA model. This parameter corresponds to the `batch_size` parameter in scikit-learn.
   - **Initial value of iteration used in learning update schedule**. Specify the starting value that downweights the learning rate for early iterations in online learning. This parameter corresponds to the `learning_offset` parameter in scikit-learn.
   - **Power applied to the iteration during updates**. Indicate the level of power applied to the iteration count in order to control learning rate during online updates. This parameter corresponds to the `learning_decay` parameter in scikit-learn.
   - **Number of passes over the data**. Specify the maximum number of times the algorithm will cycle over the data. This parameter corresponds to the `max_iter` parameter in scikit-learn.

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**, enter 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. But if the number of ngrams in the input exceeds this size, collisions may occur.

10. Submit the pipeline. The LDA component 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 component has two outputs:

- **Transformed dataset**: This output contains the input text, a specified number of discovered categories, and the scores for each text example for each category.
- **Feature topic matrix**: The leftmost column contains the extracted text feature. A column for each category contains the score for that feature in that category.

LDA transformation

This component also outputs the *LDA transformation* that applies LDA to the dataset.

You can save this transformation and reuse it for other datasets. This technique might be useful if you've trained on a large corpus and want to reuse the coefficients or categories.

To reuse this transformation, select the **Register dataset** icon in the right panel of the Latent Dirichlet Allocation component to keep the component under the **Datasets** category in the component list. Then you can connect this component to the Apply Transformation component to reuse this transformation.

Refining an LDA model or results

Typically, you can't create a single LDA model that will meet all needs. 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 determine 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?

You can often improve the accuracy of models based on LDA 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.

In the designer, you can also use R or Python libraries for text processing: Execute R Script, Execute Python Script.

Technical notes

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

Implementation details

By default, the distributions of outputs for a 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 component might return an empty topic. Most often, the cause is pseudo-random initialization of the algorithm. If this happens, you can try changing related parameters. For example, change 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 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've provided a corpus of customer reviews that includes many products. The text of reviews that have been submitted by customers over time contains many terms, some of which are used in multiple topics.

A *topic* that the LDA process identifies might represent reviews for an individual product, 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. They can refer to other products, or be general terms that apply to everything ("great", "awful"). Other terms might be noise words. However, the LDA method doesn't try 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 are used in the target domain.

After the term indexes are computed, a distance-based similarity measure compares individual rows of text to determine whether two pieces of text are similar. 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.

Component parameters

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
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.

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
N-grams	Integer	[1;10]	Required	2	Order of N-grams generated during hashing.
Normalize	Boolean	True or False	Required	true	Normalize output to probabilities. The transformed dataset will be P(topic|document) and the feature topic matrix will be P(word|topic).
Show all options	Boolean	True or False	Required	False	Presents additional parameters specific to scikit-learn online LDA.
Rho parameter	Float	[0.00001;1.0]	Applies when the **Show all options** check box is selected	0.01	Topic word prior distribution.
Alpha parameter	Float	[0.00001;1.0]	Applies when the **Show all options** check box is selected	0.01	Document topic prior distribution.
Estimated number of documents	Integer	[1;int.MaxValue]	Applies when the **Show all options** check box is selected	1000	Estimated number of documents. Corresponds to the `total_samples` parameter.
Size of the batch	Integer	[1;1024]	Applies when the **Show all options** check box 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** check box is selected	0	Initial value that downweights learning rate for early iterations. Corresponds to the `learning_offset` parameter.

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Power applied to the iteration during updates	Float	[0.0;1.0]	Applies when the **Show all options** check box is selected	0.5	Power applied to the iteration count in order to control learning rate. Corresponds to the `learning_decay` parameter.
Number of training iterations	Integer	[1;1024]	Applies when the **Show all options** check box is selected	25	Number of training iterations.
Build dictionary of ngrams	Boolean	True or False	Applies when the **Show all options** check box is *not* selected	True	Builds a dictionary of ngrams prior to computing LDA. Useful for model inspection and interpretation.
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 the number of tokens in the input exceeds this size, collisions might occur.
Number of bits to use for feature hashing.	Integer	[1;31]	Applies when the **Show all options** check box is *not* selected and **Build dictionary of ngrams** is **False**	12	Number of bits to use for feature hashing.
Build dictionary of ngrams prior to LDA	Boolean	True or False	Applies when the **Show all options** check box 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 **Show all options** check box is selected and the option **Build dictionary of ngrams** is **True**	20000	Maximum size of the dictionary. If the number of tokens in the input exceeds this size, collisions might occur.

NAME	TYPE	RANGE	OPTIONAL	DEFAULT	DESCRIPTION
Number of hash bits	Integer	[1;31]	Applies when the **Show all options** check box is selected and the option **Build dictionary of ngrams** is **False**	12	Number of bits to use during feature hashing.

Next steps

See the set of components available to Azure Machine Learning.

For a list of errors specific to the components, see Exceptions and error codes for the designer.

Score Vowpal Wabbit Model

3/28/2022 • 3 minutes to read • Edit Online

This article describes how to use the **Score Vowpal Wabbit Model** component in Azure Machine Learning designer, to generate scores for a set of input data, using an existing trained Vowpal Wabbit model.

This component provides the latest version of the Vowpal Wabbit framework, version 8.8.1. Use this component to score data using a trained model saved in the VW version 8 format.

How to configure Score Vowpal Wabbit Model

1. Add the **Score Vowpal Wabbit Model** component 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 **Datasets** category of designer's left navigation pane. However, the model must be available in Azure Machine Learning Designer.

   > **NOTE**
   > Only Vowpal Wabbit 8.8.1 models are supported; you cannot connect saved models that were trained by using other algorithms.

3. Add the test dataset and connect it to right-hand input port. If test dataset is a directory, which contains the test data file, specify the test data file name with **Name of the test data file**. If test dataset is a single file, leave **Name of the test data file** to be empty.
4. 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.

5. **Name of the test data file**: Type the name of the file that contains the input data. This argument is only used when the test dataset is a directory.
6. **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.

7. 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.

8. Select the option, **Include an extra column containing raw scores**, if you want to output raw scores together with the results.
9. Submit the pipeline.

Results

After training is complete:

- To visualize the results, right-click the output of the Score Vowpal Wabbit Model component. The output indicates a prediction score normalized from 0 to 1.
- To evaluate the results, the output dataset should contain specific score column names, which meet Evaluate Model component requirements.
  - For regression task, the dataset to evaluate must has one column, named `Regression Scored Labels`, which represents scored labels.
  - For binary classification task, the dataset to evaluate must has two columns, named `Binary Class Scored Labels`, `Binary Class Scored Probabilities`, which represent scored labels, and probabilities respectively.
  - For multi classification task, the dataset to evaluate must has one column, named `Multi Class Scored Labels`, which represents scored labels.

  Note that the results of the Score Vowpal Wabbit Model component cannot be evaluated directly. Before evaluating, the dataset should be modified according to the requirements above.

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 component.

- 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.

Next steps

See the set of components available to Azure Machine Learning.

Train Vowpal Wabbit Model

3/28/2022 • 6 minutes to read • Edit Online

This article describes how to use the **Train Vowpal Wabbit Model** component in Azure Machine Learning designer, to create a machine learning model by using Vowpal Wabbit.

To use Vowpal Wabbit for machine learning, format your input according to Vowpal Wabbit requirements, and prepare the data in the required format. Use this component to specify Vowpal Wabbit command-line arguments.

When the pipeline 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.

To incrementally train an existing model on new data, connect a saved model to the **Pre-trained Vowpal Wabbit model** input port of **Train Vowpal Wabbit Model**, and add the new data to the other input port.

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 component 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 Model

This section describes how to train a new model, and how to add new data to an existing model.

Unlike other components in designer, this component both specifies the component 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

Prepare the input data

To train a model using this component, the input dataset must consist of a single text column in one of the two supported formats: **SVMLight** 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 can be read from two kinds of datasets, file dataset or tabular dataset. Both of these datasets must

either in SVMLight or VW format. 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 Model** component to your experiment.
2. Add the training dataset and connect it to **Training data**. If training dataset is a directory, which contains the training data file, specify the training data file name with **Name of the training data file**. If training dataset is a single file, leave **Name of the training data file** to be empty.
3. 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.
4. **Name of the training data file**: Type the name of the file that contains the input data. This argument is only used when the training dataset is a directory.
5. **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.
6. **Output readable model file**: select the option if you want the component to save the readable model to the run records. This argument corresponds to the `--readable_model` parameter in the VW command line.
7. **Output inverted hash file**: select the option if you want the component to save the inverted hashing function to one file in the run records. This argument corresponds to the `--invert_hash` parameter in the VW command line.
8. Submit the pipeline.

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 Model** component in the same pipeline.
- Locate a saved model in the **Datasets** category of designer's left navigation pane, and drag it in to your pipeline.

1. Add the **Train Vowpal Wabbit Model** component to your pipeline.
2. Connect the previously trained model to the **Pre-trained Vowpal Wabbit Model** input port of the component.
3. Connect the new training data to the **Training data** input port of the component.
4. In the parameters pane of **Train Vowpal Wabbit Model**, specify the format of the new training data, and also the training data file name if the input dataset is a directory.
5. Select the **Output readable model file** and **Output inverted hash file** options if the corresponding files need to be saved in the run records.

6. Submit the pipeline.

7. Select the component and select **Register dataset** under **Outputs+logs** tab in the right pane, 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.

Results

- To generate scores from the model, use Score Vowpal Wabbit Model.

> **NOTE**
> If you need to deploy the trained model in the designer, make sure that Score Vowpal Wabbit Model instead of **Score Model** is connected to the input of Web Service Output component in the inference pipeline.

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.

Supported and unsupported parameters

This section describes support for Vowpal Wabbit command line parameters in Azure Machine Learning designer.

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 component.

- 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

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 components.

Next steps

See the set of components available to Azure Machine Learning.

Apply Image Transformation

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the Apply Image Transformation component in Azure Machine Learning designer, to modify an input image directory based on a previously specified image transformation.

You need to connect an Init Image Transformation component to specify the transformation, and then you can apply such transformation to the input image directory of the Apply Image Transformation component.

How to use Apply Image Transformation

1. Add the **Apply Image Transformation** component to your pipeline. You can find this component under *Computer Vision/Image Data Transformation* category.
2. Connect the output of **Init Image Transformation** to the left-hand input of **Apply Image Transformation**.

   > **NOTE**
   > Only image transformation generated by Init Image Transformation component is accepted to this component. For other kind of transformation, please connect it to Apply Transformation, otherwise 'InvalidTransformationDirectoryError' will be thrown.

3. Connect the image directory that you want to transform.
4. For **Mode**, specify for what purpose you use input transformation: 'For training' or 'For inference'.

   If you select **For training**, all transformation you specify in Init Image Transformation will be applied.

   If you select **For inference**, transformation like creating new samples randomly will be excluded before being applied. This is because transformation operations to create new samples randomly like 'Random horizontal flip' are used for data augmentation in training, which should be removed in inference because inference samples need to be fixed for accurate prediction and evaluation.

   > **NOTE**
   > Transformations which will be excluded in mode **For inference** are: Random resized crop, Random crop, Random horizontal flip, Random vertical flip, Random rotation, Random affine, Random grayscale, Random perspective, Random erasing.

5. To apply a image transformation to a new image directory, submit the pipeline.

Component parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Mode	Any	Mode	(Require user to specify)	For what purpose you use input transformation. You should exclude 'Random' transform operations in inference but keep them in training

Expected inputs

NAME	TYPE	DESCRIPTION
Input image transformation	TransformationDirectory	Input image transformation
Input image directory	ImageDirectory	Image directory to be transformed

Outputs

NAME	TYPE	DESCRIPTION
Output image directory	ImageDirectory	Output image directory

Next steps

See the set of components available to Azure Machine Learning.

Convert to Image Directory

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the Convert to Image Directory component to help convert image dataset to *Image Directory* data type, which is standardized data format in image-related tasks like image classification in Azure Machine Learning designer.

How to use Convert to Image Directory

1. Prepare your image dataset first.

   For supervised learning, you need to specify the label of training dataset. The image dataset file should be in following structure:

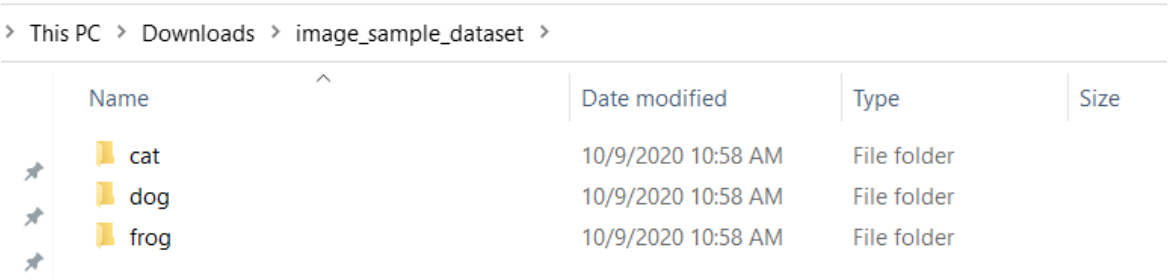
   ```
   Your_image_folder_name/Category_1/xxx.png
   Your_image_folder_name/Category_1/xxy.jpg
   Your_image_folder_name/Category_1/xxz.jpeg

   Your_image_folder_name/Category_2/123.png
   Your_image_folder_name/Category_2/nsdf3.png
   Your_image_folder_name/Category_2/asd932_.png
   ```

   In the image dataset folder, there are multiple subfolders. Each subfolder contains images of one category respectively. The names of subfolders are considered as the labels for tasks like image classification. Refer to torchvision datasets for more information.

   > **WARNING**
   > Currently labeled datasets exported from Data Labeling are not supported in the designer.

   Images with these extensions (in lowercase) are supported: '.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif', '.tiff', '.webp'. You can also have multiple types of images in one folder. It is not necessary to contain the same count of images in each category folder.

   You can either use the folder or compressed file with extension '.zip', '.tar', '.gz', and '.bz2'. **Compressed files are recommended for better performance.**

   > **NOTE**
   > For inference, the image dataset folder only needs to contain unclassified images.

2. Register the image dataset as a file dataset in your workspace, since the input of Convert to Image

Directory component must be a **File dataset**.

3. Add the registered image dataset to the canvas. You can find your registered dataset in the **Datasets** category in the component list in the left of canvas. Currently Designer does not support visualize image dataset.

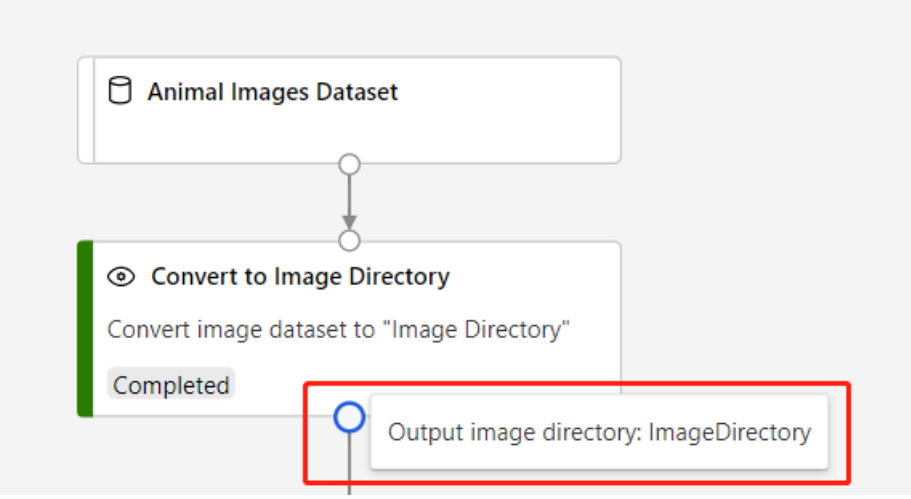
   > **WARNING**
   > You **cannot** use **Import Data** component to import image dataset, because the output type of **Import Data** component is DataFrame Directory, which only contains file path string.

4. Add the **Convert to Image Directory** component to the canvas. You can find this component in the 'Computer Vision/Image Data Transformation' category in the component list. Connect it to the image dataset.

5. Submit the pipeline. This component could be run on either GPU or CPU.

Results

The output of **Convert to Image Directory** component is in **Image Directory** format, and can be connected to other image-related components of which the input port format is also Image Directory.



Technical notes

Expected inputs

NAME	TYPE	DESCRIPTION
Input dataset	AnyDirectory, ZipFile	Input dataset

Output

NAME	TYPE	DESCRIPTION
Output image directory	ImageDirectory	Output image directory

Next steps

See the set of components available to Azure Machine Learning.

Init Image Transformation

3/28/2022 • 4 minutes to read • Edit Online

This article describes how to use the **Init Image Transformation** component in Azure Machine Learning designer, to initialize image transformation to specify how you want image to be transformed.

How to configure Init Image Transformation

1. Add the **Init Image Transformation** component to your pipeline in the designer.
2. For **Resize**, specify whether to resize the input PIL Image to the given size. If you choose 'True', you can specify the desired output image size in **Size**, by default 256.
3. For **Center crop**, specify whether to crop the given PIL Image at the center. If you choose 'True', you can specify the desired output image size of the crop in **Crop size**, by default 224.
4. For **Pad**, specify whether to pad the given PIL Image on all sides with the pad value 0. If you choose 'True', you can specify padding (how many pixels to add) on each border in **Padding**.
5. For **Color jitter**, specify whether to randomly change the brightness, contrast and saturation of an image.
6. For **Grayscale**, specify whether to convert image to grayscale.
7. For **Random resized crop**, specify whether to crop the given PIL Image to random size and aspect ratio. A crop of random size (range from 0.08 to 1.0) of the original size and a random aspect ratio (range from 3/4 to 4/3) of the original aspect ratio is made. This crop is finally resized to given size. This is commonly used in training the Inception networks. If you choose 'True', you can specify the expected output size of each edge in **Random size**, by default 256.
8. For **Random crop**, specify whether to crop the given PIL Image at a random location. If you choose 'True', you can specify the desired output size of the crop in **Random crop size**, by default 224.
9. For **Random horizontal flip**, specify whether to horizontally flip the given PIL Image randomly with probability 0.5.
10. For **Random vertical flip**, specify whether to vertically flip the given PIL Image randomly with probability 0.5.
11. For **Random rotation**, specify whether to rotate the image by angle. If you choose 'True', you can specify in range of degrees by setting **Random rotation degrees**, which means (-degrees, +degrees), by default 0.
12. For **Random affine**, specify whether to random affine transformation of the image keeping center invariant. If you choose 'True', you can specify in range of degrees to select from in **Random affine degrees**, which means (-degrees, +degrees), by default 0.
13. For **Random grayscale**, specify whether to randomly convert image to grayscale with probability 0.1.
14. For **Random perspective**, specify whether to performs Perspective transformation of the given PIL Image randomly with probability 0.5.
15. Connect to Apply Image Transformation component, to apply the transformation specified above to the input image dataset.

16. Submit the pipeline.

Results

After transformation is completed, you can find transformed images in the output of Apply Image Transformation component.

Technical notes

Refer to https://pytorch.org/vision/stable/transforms.html for more info about image transformation.

Component parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Resize	Any	Boolean	True	Resize the input PIL Image to the given size
Size	>=1	Integer	256	Specify the desired output size
Center crop	Any	Boolean	True	Crops the given PIL Image at the center
Crop size	>=1	Integer	224	Specify the desired output size of the crop
Pad	Any	Boolean	False	Pad the given PIL Image on all sides with the given "pad" value
Padding	>=0	Integer	0	Padding on each border
Color jitter	Any	Boolean	False	Randomly change the brightness, contrast and saturation of an image
Grayscale	Any	Boolean	False	Convert image to grayscale
Random resized crop	Any	Boolean	False	Crop the given PIL Image to random size and aspect ratio
Random size	>=1	Integer	256	Expected output size of each edge
Random crop	Any	Boolean	False	Crop the given PIL Image at a random location

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Random crop size	>=1	Integer	224	Desired output size of the crop
Random horizontal flip	Any	Boolean	True	Horizontally flip the given PIL Image randomly with a given probability
Random vertical flip	Any	Boolean	False	Vertically flip the given PIL Image randomly with a given probability
Random rotation	Any	Boolean	False	Rotate the image by angle
Random rotation degrees	[0,180]	Integer	0	Range of degrees to select from
Random affine	Any	Boolean	False	Random affine transformation of the image keeping center invariant
Random affine degrees	[0,180]	Integer	0	Range of degrees to select from
Random grayscale	Any	Boolean	False	Randomly convert image to grayscale with probability 0.1
Random perspective	Any	Boolean	False	Performs Perspective transformation of the given PIL Image randomly with probability 0.5
Random erasing	Any	Boolean	False	Randomly selects a rectangle region in an image and erases its pixels with probability 0.5

Output

NAME	TYPE	DESCRIPTION
Output image transformation	TransformationDirectory	Output image transformation that can be connected to **Apply Image Transformation** component.

Next steps

See the set of components available to Azure Machine Learning.

Split Image Directory

3/28/2022 • 2 minutes to read • Edit Online

This topic describes how to use the Split Image Directory component in Azure Machine Learning designer, to divide the images of an image directory into two distinct sets.

This component is particularly useful when you need to separate image data into training and testing sets.

How to configure Split Image Directory

1. Add the **Split Image Directory** component to your pipeline. You can find this component under 'Computer Vision/Image Data Transformation' category.
2. Connect it to component of which the output is image directory.
3. Input **Fraction of images in the first output** to specify the percentage of data to put in the left split, by default 0.9. If the fraction result is not integer, the component uses the smaller near integer.

Technical notes

Expected inputs

NAME	TYPE	DESCRIPTION
Input image directory	ImageDirectory	Image directory to split

Component parameters

NAME	TYPE	RANGE	OPTIONAL	DESCRIPTION	DEFAULT
Fraction of images in the first output	Float	0-1	Required	Fraction of images in the first output	0.9

Outputs

NAME	TYPE	DESCRIPTION
Output image directory1	ImageDirectory	Image directory that contains selected images
Output image directory2	ImageDirectory	Image directory that contains all other images

Next steps

See the set of components available to Azure Machine Learning.

DenseNet

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the **DenseNet** component in Azure Machine Learning designer, to create an image classification model using the Densenet algorithm.

This classification algorithm is a supervised learning method, and requires a labeled image directory.

> **NOTE**
> This component does not support labeled dataset generated from *Data Labeling* in the studio, but only support labeled image directory generated from Convert to Image Directory component.

You can train the model by providing the model and the labeled image directory as inputs to Train Pytorch Model. The trained model can then be used to predict values for the new input examples using Score Image Model.

More about DenseNet

For more information on DenseNet, see the research paper, Densely Connected Convolutional Networks.

How to configure DenseNet

1. Add the **DenseNet** component to your pipeline in the designer.
2. For **Model name**, specify name of a certain DenseNet structure and you can select from supported DenseNet: 'densenet121', 'densenet161', 'densenet169', 'densenet201'.
3. For **Pretrained**, specify whether to use a model pre-trained on ImageNet. If selected, you can fine-tune model based on selected pre-trained model; if deselected, you can train from scratch.
4. For **Memory efficient**, specify whether to use checkpointing, which is much more memory-efficient but slower. For more information, see the research paper, Memory-Efficient Implementation of DenseNets.
5. Connect the output of **DenseNet** component, training, and validation image dataset component to the Train Pytorch Model.
6. Submit the pipeline.

Results

After pipeline run is completed, to use the model for scoring, connect the Train Pytorch Model to Score Image Model, to predict values for new input examples.

Technical notes

component parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Model name	Any	Mode	densenet201	Name of a certain DenseNet structure

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Pretrained	Any	Boolean	True	Whether to use a model pre-trained on ImageNet
Memory efficient	Any	Boolean	False	Whether to use checkpointing, which is much more memory efficient but slower

Output

NAME	TYPE	DESCRIPTION
Untrained model	UntrainedModelDirectory	An untrained DenseNet model that can be connected to Train Pytorch Model.

Next steps

See the set of components available to Azure Machine Learning.

ResNet

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the **ResNet** component in Azure Machine Learning designer, to create an image classification model using the ResNet algorithm..

This classification algorithm is a supervised learning method, and requires a labeled dataset.

> **NOTE**
> This component does not support labeled dataset generated from *Data Labeling* in the studio, but only support labeled image directory generated from Convert to Image Directory component.

You can train the model by providing a model and a labeled image directory as inputs to Train Pytorch Model. The trained model can then be used to predict values for the new input examples using Score Image Model.

More about ResNet

Refer to this paper for more details about ResNet.

How to configure ResNet

1. Add the **ResNet** component to your pipeline in the designer.
2. For **Model name**, specify name of a certain ResNet structure and you can select from supported resnet: 'resnet18', 'resnet34', 'resnet50', 'resnet101', 'resnet152', 'resnet152', 'resnext50_32x4d', 'resnext101_32x8d', 'wide_resnet50_2', 'wide_resnet101_2'.
3. For **Pretrained**, specify whether to use a model pre-trained on ImageNet. If selected, you can fine-tune model based on selected pre-trained model; if deselected, you can train from scratch.
4. For **Zero init residual**, specify whether to zero-initialize the last batch norm layer in each residual branch. If selected, the residual branch starts with zeros, and each residual block behaves like an identity. This can help with convergence at large batch sizes according to https://arxiv.org/abs/1706.02677.
5. Connect the output of **ResNet** component, training and validation image dataset component to the Train Pytorch Model.
6. Submit the pipeline.

Results

After pipeline run is completed, to use the model for scoring, connect the Train PyTorch Model to Score Image Model, to predict values for new input examples.

Technical notes

Component parameters

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Model name	Any	Mode	resnext101_32x8d	Name of a certain ResNet structure

NAME	RANGE	TYPE	DEFAULT	DESCRIPTION
Pretrained	Any	Boolean	True	Whether to use a model pre-trained on ImageNet
Zero init residual	Any	Boolean	False	Whether to zero-initialize the last batch norm layer in each residual branch

Output

NAME	TYPE	DESCRIPTION
Untrained model	UntrainedModelDirectory	An untrained ResNet model that can be connected to Train Pytorch Model.

Next steps

See the set of components available to Azure Machine Learning.

Evaluate Recommender

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the Evaluate Recommender component in Azure Machine Learning designer. The goal is to measure the accuracy of predictions that a recommendation model has made. By using this component, you can evaluate different kinds of recommendations:

- Ratings predicted for a user and an item
- Items recommended for a user

When you create predictions by using a recommendation model, slightly different results are returned for each of these supported prediction types. The Evaluate Recommender component 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

The component also applies the appropriate performance metrics, based on the type of prediction being made.

How to configure Evaluate Recommender

The Evaluate Recommender component compares the prediction output by using a recommendation model with the corresponding "ground truth" data. For example, the Score SVD Recommender component produces scored datasets that you can analyze by using 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.

Scored dataset

The scored dataset contains the predictions that the recommendation model generated.

The columns in this second dataset depend on the kind of prediction that you performed during the scoring process. For example, the scored dataset might contain either of the following:

- Users, items, and the ratings that the user would likely give for the item
- A list of users and items recommended for them

Metrics

Performance metrics for the model are generated based on the type of input. The following sections give details.

Evaluate predicted ratings

When you're evaluating predicted ratings, the scored dataset (the second input to Evaluate Recommender) must contain user-item-rating triples that meet these requirements:

- The first column of the dataset contains the 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. It then computes the mean absolute error (MAE) and the root mean squared error (RMSE).

Evaluate item recommendations

When you're evaluating item recommendations, 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 you connect this dataset, we recommend that you sort the dataset so that the most relevant items come first.

> **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's 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 component must only produce recommendations for items with "ground truth" ratings (in the test dataset).

Next steps

See the set of components available to Azure Machine Learning.

Score SVD Recommender

3/28/2022 • 5 minutes to read • Edit Online

This article describes how to use the Score SVD Recommender component in Azure Machine Learning designer. Use this component to create predictions by using a trained recommendation model based on the Single Value Decomposition (SVD) algorithm.

The SVD recommender can generate two different kinds of predictions:

- Predict ratings for a given user and item
- Recommend items to a user

When you're creating the second type of predictions, you can operate in one of these modes:

- **Production mode** considers all users or items. It's typically used in a web service.

  You can create scores for new users, not just users seen during training. For more information, see the technical notes.

- **Evaluation mode** operates on a reduced set of users or items that can be evaluated. It's typically used during pipeline operations.

For more information on the SVD recommender algorithm, see the research paper Matrix factorization techniques for recommender systems.

How to configure Score SVD Recommender

This component supports two types of predictions, each with different requirements.

Prediction of ratings

When you predict ratings, the model calculates how a user will react to a particular item, given the training data. The input data for scoring must provide both a user and the item to rate.

1. Add a trained recommendation model to your pipeline, and connect it to **Trained SVD recommender**. You must create the model by using the Train SVD Recommender component.
2. For **Recommender prediction kind**, select **Rating Prediction**. No other parameters are required.
3. Add the data for which you want to make predictions, and connect it to **Dataset to score**.

   For the model 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. Submit the pipeline.

Results for rating predictions

The output dataset contains three columns: users, items, and the predicted rating for each input user and item.

Recommendations for users

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 you can set a threshold for the number of previous recommendations that are required to generate a recommendation.

1. Add a trained recommendation model to your pipeline, and connect it to **Trained SVD recommender**. You must create the model by using the Train SVD Recommender component.

2. To recommend items for a list of users, set **Recommender prediction kind** to **Item Recommendation**.

3. For **Recommended item selection**, indicate whether you're using the scoring component in production or for model evaluation. Choose one of these values:

   - **From All Items**: Select this option if you're setting up a pipeline to use in a web service or in production. This option enables *production mode*. The component makes recommendations from all items seen during training.

   - **From Rated Items (for model evaluation)**: Select this option if you're developing or testing a model. This option enables *evaluation mode*. The component makes recommendations only from those items in the input dataset that have been rated.

   - **From Unrated Items (to suggest new items to users)**: Select this option if you want the component to make recommendations only from those items in the training dataset that have not been rated.

4. Add the dataset for which you want to make predictions, and connect it to **Dataset to score**.

   - For **From All Items**, the input dataset should consist of one column. It contains the identifiers of users for which to make recommendations.

     The dataset can include an extra two columns of item identifiers and ratings, but these two columns are ignored.

   - For **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.

   - For **From Unrated Items (to suggest new items to users)**, 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. **Maximum number of items to recommend to a user**: Enter the number of items to return for each user. By default, the component recommends five items.

6. **Minimum size of the recommendation pool per user**: Enter a value that indicates how many prior recommendations are required. By default, this parameter is set to 2, meaning at least two other users have recommended the item.

   Use this option only if you're scoring in evaluation mode. The option is not available if you select **From All Items** or **From Unrated Items (to suggest new items to users)**.

7. For **From Unrated Items (to suggest new items to users)**, use the third input port, named **Training Data**, 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.

8. Submit the pipeline.

Results of item recommendation

The scored dataset returned by Score SVD 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 that 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. The item with highest affinity is put in column **Item 1**.

Technical notes

If you have a pipeline with the SVD recommender, and you move the model to production, be aware that there are key differences between using the recommender in evaluation mode and using it in production mode.

Evaluation, by definition, requires predictions that can be verified against the *ground truth* in a test set. When you evaluate the recommender, it must predict only items that have been rated in the test set. This restricts the possible values that are predicted.

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's no corresponding ground truth. So the accuracy of the recommendation can't be verified in the same way as during pipeline operations.

Next steps

See the set of components available to Azure Machine Learning.

Score Wide and Deep Recommender

3/28/2022 • 9 minutes to read • Edit Online

This article describes how to use the **Score Wide and Deep Recommender** component in Azure Machine Learning designer, to create predictions based on a trained recommendation model, based on the Wide & Deep learning from Google.

The Wide and Deep recommender can generate two different kinds of predictions:

- Predict ratings for a given user and item
- Recommend items to a given user

When creating the latter kind 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.
- **Evaluation mode** operates on a reduced set of users or items that can be evaluated, and is typically used during experimentation.

More details on the Wide and Deep recommender and its underlying theory can be found in the relevant research paper: Wide & Deep Learning for Recommender Systems.

How to configure Score Wide and Deep Recommender

This component 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

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 Wide & Deep recommendation model to your experiment, and connect it to **Trained Wide and Deep recommendation model**. You must create the model by using Train Wide and Deep 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 in the training dataset are ignored by **Score Wide and Deep 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 in the training dataset are ignored by **Score Wide and Deep 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. 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:

- For a numeric user or item feature column, missing values are automatically replaced with the **mean** of its non-missing training set values. For categorical feature, missing values are replaced with the same categorical value other than any possible values of this feature.
- No translation is applied to the feature values, to maintain their sparsity.

Recommend items

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 Wide and Deep recommendation model to your experiment, and connect it to **Trained Wide and Deep recommendation model**. You must create the model by using Train Wide and Deep 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 component 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 component 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 component makes recommendations from all items seen during training.
   - **From Unrated Items (to suggest new items to users)**: Select this option if you want the

component to make recommendations only from those items in the training dataset that have not been rated.

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.

     The dataset can include an extra two columns of item identifiers and ratings, but these two columns are ignored.

   - 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.

   - For **From Unrated Items (to suggest new items to users)**, 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 Wide and Deep 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 Wide and Deep 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** or **From Unrated Items (to suggest new items to users)**.

9. For **From Unrated Items (to suggest new items to users)**, use the third input port, named **Training Data**, 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.

10. Run the experiment.

Results of item recommendation

The scored dataset returned by **Score Wide and Deep 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**.

Technical notes

This section contains answers to some common questions about using the Wide & Deep recommender to create predictions.

Cold-start users and recommendations

Typically, to create recommendations, the **Score Wide and Deep Recommender** component 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 of using a classification or regression learner.

Production use of the Wide and Deep recommender

If you have experimented with the Wide and Deep 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.

Next steps

See the set of components available of Azure Machine Learning.

Train SVD Recommender

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the Train SVD Recommender component in Azure Machine Learning designer. Use this component to train a recommendation model based on the Single Value Decomposition (SVD) algorithm.

The Train SVD Recommender component reads a dataset of user-item-rating triples. It returns a trained SVD recommender. You can then use the trained model to predict ratings or generate recommendations, by connecting the Score SVD Recommender component.

More about recommendation models and the SVD recommender

The main aim of a recommendation system is to recommend one or more *items* to *users* of the system. Examples of an item might be a movie, restaurant, book, or song. A user might be a person, a group of persons, or another entity with item preferences.

There are two principal approaches to recommender systems:

- A **content-based** approach makes use of features for both users and items. Users can be described by properties such as age and gender. Items can be described by properties such as author and manufacturer. You can find typical examples of content-based recommendation systems on social matchmaking sites.
- **Collaborative filtering** uses only identifiers of the users and the items. It gets 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've rated and from other users who have rated the same items.

The SVD recommender uses identifiers of the users and the items, and a matrix of ratings given by the users to the items. It's a *collaborative recommender*.

For more information about the SVD recommender, see the relevant research paper: Matrix factorization techniques for recommender systems.

How to configure Train SVD Recommender

Prepare data

Before you use the component, your input data must be in the format that the recommendation model expects. A training data set of user-item-rating triples is required.

- The first column contains user identifiers.
- The second column contains item identifiers.
- The third column contains the rating for the user-item pair. Rating values must be numeric type.

The **Movie Ratings** dataset in Azure Machine Learning designer (select **Datasets** and then **Samples**) demonstrates the expected format:

Movie Ratings result visualization

Rows 227,472 | Columns (up to 100 columns/rows could be visualized) 4

UserId	MovieId	Rating	Timestamp
1	68646	10	1381620027
1	113277	10	1379466669
2	454876	8	1394818630
2	790636	7	1389963947
2	816711	8	1379963769
2	1091191	7	1391173869
2	1322269	7	1391529691
2	1433811	8	1380453043
2	1454468	8	1387016442
2	1535109	8	1386350135
2	1675434	8	1396688981
2	1798709	10	1389948338
2	2017038	7	1390570895
2	2024544	10	1389355949

From this sample, you can see that a single user has rated several movies.

Train the model

1. Add the Train SVD Recommender component to your pipeline in the designer, and connect it to the training data.
2. For **Number of factors**, specify the number of factors to use with the recommender.

   Each factor measures how much the user is relating with the item. The number of factors is also the dimensionality of latent factor space. With the number of users and items increasing, it's better to set a larger number of factors. But if the number is too large, performance might drop.
3. **Number of recommendation algorithm iterations** indicates how many times the algorithm should process the input data. The higher this number is, the more accurate the predictions are. However, a higher number means slower training. The default value is 30.
4. For **Learning rate**, enter a number between 0.0 and 2.0 that defines the step size for learning.

   The learning rate determines the size of the step at each iteration. If the step size is too large, you might overshoot the optimal solution. If the step size is too small, training takes longer to find the best solution.
5. Submit the pipeline.

Results

After pipeline run is completed, to use the model for scoring, connect the Train SVD Recommender to Score SVD Recommender, to predict values for new input examples.

Next steps

See the set of components available to Azure Machine Learning.

Train Wide & Deep Recommender

3/28/2022 • 7 minutes to read • Edit Online

This article describes how to use the **Train Wide & Deep Recommender** component in Azure Machine Learning designer, to train a recommendation model. This component is based on Wide & Deep learning, which is proposed by Google.

The **Train Wide & Deep Recommender** component reads a dataset of user-item-rating triples and, optionally, some user and item features. It returns a trained Wide & Deep recommender. You can then use the trained model to generate rating predictions or recommendations by using the Score Wide and Deep Recommender component.

More about recommendation models and the Wide & Deep 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 Wide & Deep 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, Wide & Deep recommender will still work in its collaborative filtering mode.

More details on the Wide & Deep recommender and the underlying probabilistic algorithm can be found in the relevant research paper: Wide & Deep Learning for Recommender Systems.

How to configure Train Wide & Deep Recommender

- Prepare the training data
- Train the model

Prepare data

Before trying to use the component, make sure your data is in the expected format for 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.

Required dataset of user-item-ratings

The input data used for training must 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 numeric type.

For example, a typical set of user-item-ratings might look like this:

USERID	MOVIEID	RATING
1	68646	10
223	31381	10

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, a typical set of user features might look like this:

USERID	AGE	GENDER	INTEREST	LOCATION
1	25	male	Drama	Europe
223	40	female	Romance	Asia

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, a typical set of item features might look like this:

MOVIEID	TITLE	ORIGINAL LANGUAGE	GENRES	YEAR
68646	The Godfather	English	Drama	1972
31381	Gone with the Wind	English	History	1939

Train the model

1. Add the **Train Wide and Deep Recommender** component to your experiment in the designer, and connect it to the training dataset.
2. If you have a separate dataset of either user features and/or item features, connect them to the **Train Wide and Deep Recommender** component.
   - **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. **Epochs**: indicate how many times the algorithm should process the whole training data.

   The higher this number, the more adequate the training; however, training costs more time and may

cause overfitting.

4. **Batch size**: type the number of training examples utilized in one training step.

   This hyperparameter can influence the training speed. A higher batch size leads to a less time cost epoch, but may increase the convergence time. And if batch is too large to fit GPU/CPU, a memory error may raised.

5. **Wide part optimizer**: select one optimizer to apply gradients to the wide part of the model.

6. **Wide optimizer learning rate**: enter a number between 0.0 and 2.0 that defines the learning rate of wide part optimizer.

   This hyperparameter determines the step size at each training step while moving toward a minimum of loss function. A large learning rate may cause learning jump over the minima, while a too small learning rate may cause convergence problem.

7. **Crossed feature dimension**: type the dimension by entering the desired user IDs and item ID features.

   The Wide & Deep recommender performs cross-product transformation over user ID and item ID features by default. The crossed result will be hashed according to this number to ensure the dimension.

8. **Deep part optimizer**: select one optimizer to apply gradients to the deep part of the model.

9. **Deep optimizer learning rate**: enter a number between 0.0 and 2.0 that defines the learning rate of deep part optimizer.

10. **User embedding dimension**: type an integer to specify the dimension of user ID embedding.

    The Wide & Deep recommender creates the shared user ID embeddings and item ID embeddings for both wide part and deep part.

11. **Item embedding dimension**: type an integer to specify the dimension of item ID embedding.

12. **Categorical features embedding dimension**: enter an integer to specify the dimensions of categorical feature embeddings.

    In deep component of Wide & Deep recommender, an embedding vector is learnt for each categorical feature. And these embedding vectors share the same dimension.

13. **Hidden units**: type the number of hidden nodes of deep component. The nodes number in each layer is separated by commas. For example, by type "1000,500,100", you specify the deep component has three layers, with the first layer to the last respectively has 1000 nodes, 500 nodes, and 100 nodes.

14. **Activation function**: select one activation function applied to each layer, the default is ReLU.

15. **Dropout**: enter a number between 0.0 and 1.0 to determine the probability the outputs will be dropped in each layer during training.

    Dropout is a regularization method to prevent neural networks from overfitting. One common decision for this value is to start with 0.5, which seems to be close to optimal for a wide range of networks and tasks.

16. **Batch Normalization**: select this option to use batch normalization after each hidden layer in the deep component.

    Batch normalization is a technique to fight internal covariate shift problem during networks training. In general, it can help to improve the speed, performance and stability of the networks.

17. Run the pipeline.

Technical notes

The Wide & Deep jointly trains wide linear models and deep neural networks to combine the strengths of memorization and generalization. The wide component accepts a set of raw features and feature transformations to memorize feature interactions. And with less feature engineering, the deep component generalizes to unseen feature combinations through low-dimensional dense feature embeddings.

In the implementation of Wide & Deep recommender, the component uses a default model structure. The wide component takes user embeddings, item embeddings, and the cross-product transformation of user IDs and item IDs as input. For the deep part of the model, an embedding vector is learnt for each categorical feature. Together with other numeric feature vectors, these vectors are then fed into the deep feed-forward neural network. The wide part and deep part are combined by summing up their final output log odds as the prediction, which finally goes to one common loss function for joint training.

Next steps

See the set of components available of Azure Machine Learning.

PCA-Based Anomaly Detection component

3/28/2022 • 5 minutes to read • Edit Online

This article describes how to use the PCA-Based Anomaly Detection component in Azure Machine Learning designer, to create an anomaly detection model based on principal component analysis (PCA).

This component helps you build a model in scenarios where it's easy to get training data from one class, such as valid transactions, but difficult to get sufficient samples of the targeted anomalies.

For example, to detect fraudulent transactions, you often don't have enough examples of fraud to train on. But you might have many examples of good transactions. The PCA-Based Anomaly Detection component solves the problem by analyzing available features to determine what constitutes a "normal" class. The component then applies distance metrics to identify cases that represent anomalies. This approach lets you train a model by using existing imbalanced data.

More about principal component analysis

PCA is an established technique in machine learning. It's 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. 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 more information about how PCA works, and about the implementation for anomaly detection, see these papers:

- A randomized algorithm for principal component analysis, by Rokhlin, Szlan, and Tygert
- Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions (PDF download), by Halko, Martinsson, and Tropp

How to configure PCA-Based Anomaly Detection

1. Add the **PCA-Based Anomaly Detection** component to your pipeline in the designer. You can find this component in the **Anomaly Detection** category.
2. In the right panel of the component, select the **Training mode** option. Indicate whether you want to train the model by using a specific set of parameters, or use a parameter sweep to find the best parameters.

   If you know how you want to configure the model, select the **Single Parameter** option, and provide a specific set of values as arguments.
3. For **Number of components to use in PCA**, specify the number of output features or components that you want.

   The decision of how many components to include is an important part of experiment design that uses PCA. General guidance is that you should not include the same number of PCA components as there are variables. Instead, you should start with a smaller number of components and increase them until some criterion is met.

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're 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. (This option is available when you're using the **Single parameter** training method.)

   > **NOTE**
   > You can't view the oversampled data set. For more information on how oversampling is used with PCA, see Technical notes.

5. Select the **Enable input feature mean normalization** 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 it if values have already been normalized through a different method or scale.

6. Connect a tagged training dataset and one of the training components.

   If you set the **Create trainer mode** option to **Single Parameter**, use the Train Anomaly Detection Model component.

7. Submit the pipeline.

Results

When training is complete, you can save the trained model. Or you can connect it to the Score Model component to predict anomaly scores.

To evaluate the results of an anomaly detection model:

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," you're using a typical evaluation dataset that contains a label column but no probability scores. Choose a dataset that matches the schema output for anomaly detection models, which includes **Scored Labels** and **Scored Probabilities** columns.

2. Ensure that label columns are marked.

   Sometimes the metadata associated with the label column is removed in the pipeline graph. If this happens, when you use the Evaluate Model component to compare the results of two anomaly detection models, you might get the error "There is no label column in scored dataset." Or you might get the error "There is no label column in scored dataset to compare."

   You can avoid these errors by adding the Edit Metadata component before the Evaluate Model component. Use the column selector to choose the class column, and in the **Fields** list, select **Label**.

3. Use the Execute Python Script component to adjust label column categories as **1(positive, normal)** and **0(negative, abnormal)**.

```
label_column_name = 'XXX'
anomaly_label_category = YY
dataframe1[label_column_name] = dataframe1[label_column_name].apply(lambda x: 0 if x ==
anomaly_label_category else 1)
```

Technical notes

This algorithm uses PCA to approximate the subspace that contains 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.

Next steps

See the set of components available to Azure Machine Learning.

See Exceptions and error codes for the designer for a list of errors specific to the designer components.

Train Anomaly Detection Model component

3/28/2022 • 2 minutes to read • Edit Online

This article describes how to use the Train Anomaly Detection Model component in Azure Machine Learning designer to create a trained anomaly detection model.

The component takes as input a set of parameters for an anomaly detection model 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 the designer, see PCA-Based Anomaly Detection.

How to configure Train Anomaly Detection Model

1. Add the **Train Anomaly Detection Model** component to your pipeline in the designer. You can find this component in the **Anomaly Detection** category.
2. Connect one of the components designed for anomaly detection, such as PCA-Based Anomaly Detection.

   Other types of models are not supported. When you run the pipeline, you'll get the error "All models must have the same learner type."
3. Configure the anomaly detection component by choosing the label column and setting other parameters specific to the algorithm.
4. Attach a training dataset to the right-side input of **Train Anomaly Detection Model**.
5. Submit the pipeline.

Results

After training is complete:

- To view the model's parameters, right-click the component and select **Visualize**.
- To create predictions, use the Score Model component with new input data.
- To save a snapshot of the trained model, select the component. Then select the **Register dataset** icon under the **Outputs+logs** tab in the right panel.

Next steps

See the set of components available to Azure Machine Learning.

See Exceptions and error codes for the designer for a list of errors specific to the designer components.

Web Service Input and Web Service Output components

3/28/2022 • 2 minutes to read • Edit Online

This article describes the Web Service Input and Web Service Output components in Azure Machine Learning designer.

The Web Service Input component can only connect to an input port with the type **DataFrameDirectory**. The Web Service Output component can only be connected from an output port with the type **DataFrameDirectory**. You can find the two components in the component tree, under the **Web Service** category.

The Web Service Input component indicates where user data enters the pipeline. The Web Service Output component indicates where user data is returned in a real-time inference pipeline.

How to use Web Service Input and Output

When you create a real-time inference pipeline from your training pipeline, the Web Service Input and Web Service Output components will be automatically added to show where user data enters the pipeline and where data is returned.

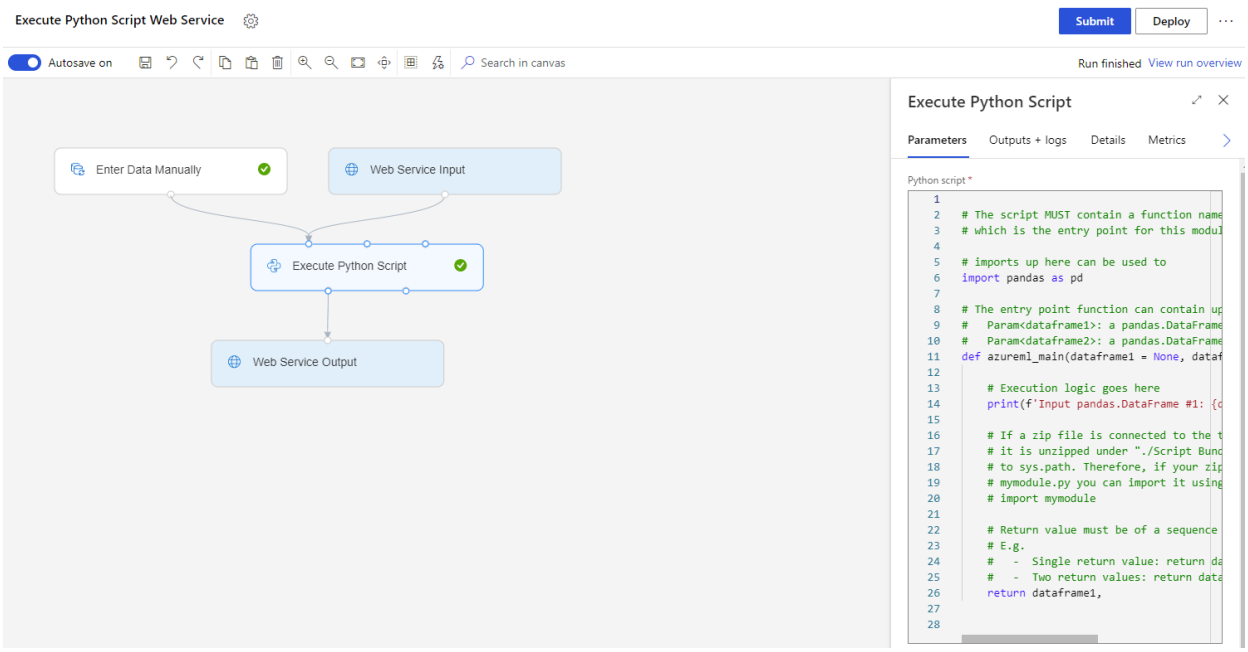
> **NOTE**
> Automatically generating a real-time inference pipeline is a rule-based, best-effort process. There's no guarantee of correctness.

You can manually add or remove the Web Service Input and Web Service Output components to satisfy your requirements. Make sure that your real-time inference pipeline has at least one Web Service Input component and one Web Service Output component. If you have multiple Web Service Input or Web Service Output components, make sure they have unique names. You can enter the name in the right panel of the component.

You can also manually create a real-time inference pipeline by adding Web Service Input and Web Service Output components to your unsubmitted pipeline.

> **NOTE**
> The pipeline type will be determined the first time you submit it. Be sure to add Web Service Input and Web Service Output components before you submit for the first time.

The following example shows how to manually create real-time inference pipeline from the Execute Python Script component.



After you submit the pipeline and the run finishes successfully, you can deploy the real-time endpoint.

> **NOTE**
>
> In the preceding example, **Enter Data Manually** provides the data schema for web service input and is necessary for deploying the real-time endpoint. Generally, you should always connect a component or dataset to the port where **Web Service Input** is connected to provide the data schema.

Next steps

Learn more about deploying the real-time endpoint.

See the set of components available to Azure Machine Learning.

Exceptions and error codes for the designer

3/28/2022 • 59 minutes to read • Edit Online

This article describes the error messages and exception codes in Azure Machine Learning designer to help you troubleshoot your machine learning pipelines.

You can find the error message in the designer following these steps:

- Select the failed component, go to the **Outputs+logs** tab, you can find the detailed log in the **70_driver_log.txt** file under the **azureml-logs** category.
- For detailed component error, you can check it in the error_info.json under **module_statistics** category.

Following are error codes of components in the designer.

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 component, 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 pipeline.

Resolution: Revisit the component 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 "{column_id}" not found.
Column with name or index "{column_id}" does not exist in "{arg_name_missing_column}".
Column with name or index "{column_id}" does not exist in "{arg_name_missing_column}", but exists in "{arg_name_has_column}".
Columns with name or index "{column_names}" not found.
Columns with name or index "{column_names}" does not exist in "{arg_name_missing_column}".
Columns with name or index "{column_names}" does not exist in "{arg_name_missing_column}", but exists in "{arg_name_has_column}".

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 component requirements and determine which value type is required (string, integer,

double, etc.)

EXCEPTION MESSAGES
Failed to parse parameter.
Failed to parse "{arg_name_or_column}" parameter.
Failed to convert "{arg_name_or_column}" parameter to "{to_type}".
Failed to convert "{arg_name_or_column}" parameter from "{from_type}" to "{to_type}".
Failed to convert "{arg_name_or_column}" parameter value "{arg_value}" from "{from_type}" to "{to_type}".
Failed to convert value "{arg_value}" in column "{arg_name_or_column}" from "{from_type}" to "{to_type}" with usage of the format "{fmt}" 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 component 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 component 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 "{name}" 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 component to process the data.

Resolution: Revisit the component throwing the exception and modify the parameter to be greater than the specified value.

EXCEPTION MESSAGES

EXCEPTION MESSAGES
Parameter should be greater than boundary value.
Parameter "{arg_name}" value should be greater than {lower_boundary}.
Parameter "{arg_name}" has value "{actual_value}" which should be greater than {lower_boundary}.

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 component to process the data.

Resolution: Revisit the component 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 "{arg_name}" value should be greater than or equal to {lower_boundary}.
Parameter "{arg_name}" has value "{value}" which should be greater than or equal to {lower_boundary}.

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 component to process the data.

Resolution: Revisit the component 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 "{arg_name}" value should be less than parameter "{upper_boundary_parameter_name}" value.
Parameter "{arg_name}" has value "{value}" which should be less than {upper_boundary_parameter_name}.

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 component, 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 component 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 component takes multiple inputs, ensure that inputs are of the same size.
- 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 "{arg_name}" value should be less than or equal to parameter "{upper_boundary_parameter_name}" value.
Parameter "{arg_name}" has value "{actual_value}" which should be less than or equal to {upper_boundary}.
Parameter "{arg_name}" value {actual_value} should be less than or equal to parameter "{upper_boundary_parameter_name}" value {upper_boundary}.
Parameter "{arg_name}" value {actual_value} should be less than or equal to {upper_boundary_meaning} value {upper_boundary}.

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 component 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 component throwing the exception and modify the parameter to be within the specified range.

EXCEPTION MESSAGES
Parameter value is not in the specified range.
Parameter "{arg_name}" value is not in range.
Parameter "{arg_name}" value should be in the range of [{lower_boundary}, {upper_boundary}].
Parameter "{arg_name}" value is not in range. {reason}

Error 0009

Exception occurs when the Azure storage account name or container name is specified incorrectly.

This error occurs in Azure Machine Learning designer 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 designer. 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 component, 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**.

EXCEPTION MESSAGES
The Azure storage account name or container name is incorrect.
The Azure storage account name "{account_name}" or container name "{container_name}" 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 {col_index} (zero-based) of input datasets ({dataset1} and {dataset2} 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 component to work.

Resolution: Modify the column selection in the component so that it will apply to the columns in the dataset.

If the component 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 pipeline.

EXCEPTION MESSAGES
Specified column set does not apply to any of dataset columns.
Specified column set "{column_set}" 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, please train model first.
Untrained model ({arg_name}), use trained model.

Error 0013

Exception occurs if the learner passed to the component is an invalid type.

This error occurs whenever a trained model is incompatible with the connected scoring component.

Resolution:

Determine the type of learner that is produced by the training component, and determine the scoring component that is appropriate for the learner.

If the model was trained using any of the specialized training components, connect the trained model only to the corresponding specialized scoring component.

MODEL TYPE	TRAINING COMPONENT	SCORING COMPONENT
any classifier	Train Model	Score Model
any regression model	Train Model	Score Model

EXCEPTION MESSAGES
Learner of invalid type is passed.

EXCEPTION MESSAGES
Learner "{arg_name}" has invalid type.
Learner "{arg_name}" has invalid type "{learner_type}".
Learner of invalid type is passed. Exception message: {exception_message}

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, like an ID column or text column. 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.

The error of unique values is greater than allowed will occur if meeting **both** following conditions:

- More than 97% instances of one column are unique values, which means nearly all categories are different from each other.
- One column has more than than 1000 unique values.

Resolution:

Open the component that generated the error, and identify the columns used as inputs. For some components, 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 ID columns which is not meaningful features during training a model, you can use Edit Metadata to mark that column as **Clear feature** and it will not be used during training a model.

For text columns, you can use Feature Hashing or Extract N-Gram Features from Text component to preprocess text columns.

> **TIP**
> Unable to find a resolution that matches your scenario? You can provide feedback on this topic that includes the name of the component 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
Amount of column unique values is greater than allowed.
Number of unique values in column: "{column_name}" is greater than allowed.
Number of unique values in column: "{column_name}" exceeds tuple count of {limitation}.

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: {connection_str}.

Error 0016

Exception occurs if input datasets passed to the component 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 or modify the original input 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 '{first_col_names}' are incompatible between train and test data.
Columns '{first_col_names}' and '{second_col_names}' are incompatible.
Column element types are not compatible for column '{first_col_names}' (zero-based) of input datasets ({first_dataset_names} and {second_dataset_names} respectively).

Error 0017

Exception occurs if a selected column uses a data type that is not supported by the current component.

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 component, 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 component.
3. Modify the column to make it conform to requirements. You might need to use several of the following components 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.
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 component 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 component.
Cannot process column of type {col_type}. The type is not supported by the component.
Cannot process column "{col_name}" of type {col_type}. The type is not supported by the component.
Cannot process column "{col_name}" of type {col_type}. The type is not supported by the component. Parameter name: {arg_name}.

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 component has the wrong number of columns, or that the data type does not match requirements of the component. For example:

- The component requires a label column, but no column is marked as a label, or you have not selected a label column yet.
- The component requires that data be categorical but your data is numeric.
- 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 component that will be consuming the dataset as input.

.

EXCEPTION MESSAGES
Dataset is not valid.
{dataset1} contains invalid data.
{dataset1} and {dataset2} should be consistent columnwise.
{dataset1} contains invalid data, {reason}.
{dataset1} contains {invalid_data_category}. {troubleshoot_hint}
{dataset1} is not valid, {reason}. {troubleshoot_hint}

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 component.

EXCEPTION MESSAGES
Values in column are not sorted.
Values in column "{col_index}" are not sorted.
Values in column "{col_index}" of dataset "{dataset}" are not sorted.
Values in argument "{arg_name}" are not sorted in "{sorting_order}" order.

Error 0020

Exception occurs if number of columns in some of the datasets passed to the component is too small.

You will receive this error in Azure Machine Learning if not enough columns have been selected for a component.

Resolution: Revisit the component 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 "{arg_name}" is less than allowed minimum.
Number of columns in input dataset is less than allowed minimum of {required_columns_count} column(s).
Number of columns in input dataset "{arg_name}" is less than allowed minimum of {required_columns_count} column(s).

Error 0021

Exception occurs if number of rows in some of the datasets passed to the component is too small.

This error in 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 component that returned the error, and check the input dataset and component properties.
- Verify that the input dataset is not empty and that there are enough rows of data to meet the requirements described in component 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 component 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.

EXCEPTION MESSAGES
Number of rows in input dataset is less than allowed minimum.
Number of rows in input dataset is less than allowed minimum of {required_rows_count} row(s).
Number of rows in input dataset is less than allowed minimum of {required_rows_count} row(s). {reason}
Number of rows in input dataset "{arg_name}" is less than allowed minimum of {required_rows_count} row(s).
Number of rows in input dataset "{arg_name}" is {actual_rows_count}, less than allowed minimum of {required_rows_count} row(s).
Number of "{row_type}" rows in input dataset "{arg_name}" is {actual_rows_count}, less than allowed minimum of {required_rows_count} row(s).

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 component 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 component.
- 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 component and check the column selection to ensure that the correct number of columns is selected. Verify the outputs of upstream components, 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.

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 {expected_col_count}.

EXCEPTION MESSAGES
Column selection pattern "{selection_pattern_friendly_name}" provides number of selected columns in input dataset not equal to {expected_col_count}.
Column selection pattern "{selection_pattern_friendly_name}" is expected to provide {expected_col_count} column(s) selected in input dataset, but {selected_col_count} column(s) is/are actually provided.

Error 0023

Exception occurs if target column of input dataset is not valid for the current trainer component.

This error in Azure Machine Learning occurs if the target column (as selected in the component parameters) is not of the valid data-type, contained all missing values, or was not categorical as expected.

Resolution: Revisit the component input to inspect the content of the label/target column. Make sure it does not have all missing values. If the component 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 "{column_index}".
Input dataset has unsupported target column "{column_index}" for learner of type {learner_type}.

Error 0024

Exception occurs if dataset does not contain a label column.

This error in Azure Machine Learning occurs when the component 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 component 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 component.
- To explicitly mark the column as the label column, add the Edit Metadata component 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.

EXCEPTION MESSAGES
There is no label column in "{dataset_name}".

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 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 component.

EXCEPTION MESSAGES
There is no score column in dataset.
There is no score column in "{dataset_name}".
There is no score column in "{dataset_name}" that is produced by a "{learner_type}". 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 component between the input dataset and the component. 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 component.
Equal column names in arguments "{arg_name_1}" and "{arg_name_2}" are not allowed. Please 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 "{friendly_name1}" is inconsistent with size of "{friendly_name2}".

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 component raising the error. Use the Column Selector in Edit Metadata to select columns to rename, and type the new columns 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 "{duplicated_name}" is duplicated.
The name "{duplicated_name}" is duplicated in "{arg_name}".
The name "{duplicated_name}" is duplicated. Details: {details}

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:

- 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 component 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.

EXCEPTION MESSAGES
The Uri "{invalid_url}" is invalid.

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: {file_url}.

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.
At least {required_columns_count} column(s) should be specified for input argument "{arg_name}".
At least {required_columns_count} column(s) should be specified for input argument "{arg_name}". The actual number of specified columns is {input_columns_count}.

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.
"{arg_name}" 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 must be finite.
"{arg_name}" is not finite.
Column "{column_name}" contains infinite values.

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 {user} and item {item} in rating prediction data table.
More than one rating for user {user} and item {item} in {dataset}.

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. 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.

TIP

Resolution not applicable to your case? You are welcome to send feedback on this article and provide information about the scenario, including the component 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 {required_feature_name} 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.

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.
Multiple label columns are specified in "{dataset_name}".

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, Microsoft Q&A question page for send feedback and provide information about the components that generated the error and related conditions.

EXCEPTION MESSAGES
Operation failed.
Error while completing operation: "{failed_operation}".
Error while completing operation: "{failed_operation}". Reason: "{reason}".

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 component 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 component 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 Execute Python Script components 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:
  - Clean Missing Data
  - Normalize Data

> **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 component 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 {type1} to column of type {type2}.
Could not convert column "{col_name1}" of type {type1} to column of type {type2}.
Could not convert column "{col_name1}" of type {type1} to column "{col_name2}" of type {type2}.

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 "{column_name}" -- all the elements are null references.
Cannot derive element type for column "{column_name}" of dataset "{dataset_name}" -- 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 "{column_id}" of mixed element types: Type of data[{row_1}, {column_id}] is "{type_1}". Type of data[{row_2}, {column_id}] is "{type_2}".
Cannot create column with id "{column_id}" of mixed element types: Type in chunk {chunk_id_1} is "{type_1}". Type in chunk {chunk_id_2} is "{type_2}" with chunk size: {chunk_size}.

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 component and verify that the directory path is correctly formatted and that it is accessible with the current credentials.

EXCEPTION MESSAGES
Please specify a valid output directory.
Directory: {path} cannot be created. Please specify valid path.

Error 0047

Exception occurs if number of feature columns in some of the datasets passed to the component 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 {required_columns_count} column(s).
Number of feature columns in input dataset "{arg_name}" is less than allowed minimum of {required_columns_count} 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 component 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.

EXCEPTION MESSAGES

EXCEPTION MESSAGES
Unable to open a file.
Error while opening the file: {file_name}.
Error while opening the file: {file_name}. Storage exception message: {exception}.

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 component does not match the actual format of the file, or if the file contains an unrecognizable character.

Resolution: Revisit the component 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_format} file.
Error while parsing the {file_format} file: {file_name}.
Error while parsing the {file_format} file. Reason: {failure_reason}.
Error while parsing the {file_format} file: {file_name}. Reason: {failure_reason}.

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 component 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 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 component you are using requires that the column also be marked as a feature, label, or categorical 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 component 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 "{col_name}" 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 component or other component to save results of a pipeline 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 component raises an error if you attempt to write a dataset to a blob that already exists.

- Open the component 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.

EXCEPTION MESSAGES
File or Blob "{file_path}" 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 component.

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 component and ensure that the label column is marked as a label.

EXCEPTION MESSAGES
The label column values and scored label column values are not comparable.
The label column is not as expected in "{dataset_name}".
The label column is not as expected in "{dataset_name}", {reason}.
The label column "{column_name}" is not expected in "{dataset_name}".
The label column "{column_name}" is not expected in "{dataset_name}", {reason}.

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 "{column_index_or_range}" 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 "{column_range}" 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.
Columns in chunk "{chunk_id_1}" is different with chunk "{chunk_id_2}" with chunk size: {chunk_size}.
Column count in file "{filename_1}" (count={column_count_1}) is different with file "{filename_2}" (count={column_count_2}).

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.
Got incompatible learner type: "{actual_learner_type}". Expected learner types are: "{expected_learner_type_list}".

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 "{account_name}" 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.
- 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 component 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 name is incorrect.
The Azure storage blob name "{blob_name}" is incorrect.
The Azure storage blob name with prefix "{blob_name_prefix}" does not exist.
Failed to find any Azure storage blobs under container "{container_name}".
Failed to find any Azure storage blobs with wildcard path "{blob_wildcard_path}".

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. Both are saved to the same Azure storage account as the account containing the input file.

Resolution: Revisit the component. 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 "{source_path}" could not be uploaded to Azure storage as "{dest_path}".

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 component expects during execution.

Resolution: Modify the input dataset or the parameters.

EXCEPTION MESSAGES
Unexpected number of columns in the datatable.
Unexpected number of columns in the dataset "{dataset_name}".
Expected "{expected_column_count}" column(s) but found "{actual_column_count}" column(s) instead.
In input dataset "{dataset_name}", expected "{expected_column_count}" column(s) but found "{actual_column_count}" column(s) 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 component 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 a Hadoop cluster using Hive queries
- Hive for SQL Users Cheat Sheet (PDF)

EXCEPTION MESSAGES
Hive script is incorrect.

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 component 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 component 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 "{sql_query}" is not correct.
SQL query "{sql_query}" is not correct. Exception message: {exception}.

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 component to verify that the name of the table is correct.

EXCEPTION MESSAGES
Azure table does not exist.
Azure table "{table_name}" does not exist.

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 component 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 {target_type}.

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.

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 components 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 write mode.
Unsupported blob write mode: {blob_write_mode}.

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 component 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 "{container_name}" 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 component.

This error in Azure Machine Learning is produced when one or more of the columns consumed by the component contains all missing values. For example, if a component is computing aggregate statistics for each column, it cannot operate on a column containing no data. In such cases, component 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.

EXCEPTION MESSAGES
Column {col_index_or_name} has all values missing.

Error 0081

Exception occurs in PCA component 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. It is typical in applications of PCA.

EXCEPTION MESSAGES
For dataset containing sparse feature columns number of dimensions to reduce to should be less than number of feature columns.

Error 0082

Exception occurs when a model cannot be successfully deserialized.

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 pipeline that produced the model or transform must be rerun and the model or transform must be resaved.

EXCEPTION MESSAGES
Model could not be deserialized 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.
{data_name} contains invalid data for training.
{data_name} contains invalid data for training. Learner type: {learner_type}.

EXCEPTION MESSAGES
{data_name} contains invalid data for training. Learner type: {learner_type}. Reason: {reason}.
Failed to apply "{action_name}" action on training data {data_name}. Reason: {reason}.

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 components for evaluating a model with output from an R script that contains scores.

Resolution:

EXCEPTION MESSAGES
Evaluating scores produced by Custom Model 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, please view the output log for more information: ---------- Start of error message from {script_language} interpreter ---------- {message} ---------- End of error message from {script_language} interpreter ----------

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 component properties.

EXCEPTION MESSAGES
The Hive table could not be created. For a HDInsight cluster, please ensure the Azure storage account name associated with cluster is the same as what is passed in through the component parameter.
The Hive table "{table_name}" could not be created. For a HDInsight cluster, please ensure the Azure storage account name associated with cluster is the same as what is passed in through the component parameter.

EXCEPTION MESSAGES
The Hive table "{table_name}" could not be created. For a HDInsight cluster, ensure the Azure storage account name associated with cluster is "{cluster_name}".

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.

Try reading the desired files to a new compressed zipped folder and try to add the custom component again.

EXCEPTION MESSAGES
Given ZIP file is not in the correct format.

Error 0105

This error is displayed when a component definition file contains an unsupported parameter type

This error in Azure Machine Learning is produced when the you create a custom component 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 component xml definition file is a supported type.

EXCEPTION MESSAGES
Unsupported parameter type.
Unsupported parameter type '{0}' specified.

Error 0107

Thrown when a component 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 component xml definition does not match a supported type.

Resolution: Make sure that the type property of an Output element in the custom component xml definition file is a supported type.

EXCEPTION MESSAGES
Unsupported output type.
Unsupported output type '{output_type}' specified.

Error 0125

Thrown when schema for multiple datasets does not match.

Resolution:

EXCEPTION MESSAGES
Dataset schema does not match.

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.

EXCEPTION MESSAGES
Image pixel size exceeds allowed limit.
Image pixel size in the file '{file_path}' exceeds allowed limit: '{size_limit}'.

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 '{column_name_or_index_1}' and '{column_name_or_index_2}' 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 '{dataset_name}' exceeds allowed.
Number of columns in the dataset in '{dataset_name}' exceeds allowed limit of '{component_name}'.
Number of columns in the dataset in '{dataset_name}' exceeds allowed '{limit_columns_count}' limit of '{component_name}'.

Error 0134

Exception occurs when label column is missing or has insufficient number of labeled rows.

This error occurs when the component 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** component 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 component to mark it as a label column.

Then, you can use the Clean Missing Data component 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 {required_rows_count} labeled rows.
Exception occurs when label column in dataset {dataset_name} is missing or has less than {required_rows_count} labeled rows.

Error 0138

Memory has been exhausted, unable to complete running of component. Downsampling the dataset may help to alleviate the problem.

This error occurs when the component 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.

EXCEPTION MESSAGES
Memory has been exhausted, unable to complete running of component.
Memory has been exhausted, unable to complete running of component. Details: {details}

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 {actual_num}) should be at least {lower_boundary}.

Error 0154

Exception occurs when user tries to join data on key columns with incompatible column type.

EXCEPTION MESSAGES
Key column element types are not compatible.
Key column element types are not compatible.(left: {keys_left}; right: {keys_right})

Error 0155

Exception occurs when column names of dataset are not string.

EXCEPTION MESSAGES
The dataframe column name must be string type. Column names are not string.
The dataframe column name must be string type. Column names {column_names} are not string.

Error 0156

Exception occurs when failed to read data from Azure SQL Database.

EXCEPTION MESSAGES
Failed to read data from Azure SQL Database.
Failed to read data from Azure SQL Database: {detailed_message} DB: {database_server_name}:{database_name} Query: {sql_statement}

Error 0157

Datastore not found.

EXCEPTION MESSAGES
Datastore information is invalid.
Datastore information is invalid. Failed to get AzureML datastore '{datastore_name}' in workspace '{workspace_name}'.

Error 0158

Thrown when a transformation directory is invalid.

EXCEPTION MESSAGES
Given TransformationDirectory is invalid.
TransformationDirectory "{arg_name}" is invalid. Reason: {reason}. Rerun training experiment, which generates the Transform file. If training experiment was deleted, please recreate and save the Transform file.
TransformationDirectory "{arg_name}" is invalid. Reason: {reason}. {troubleshoot_hint}

Error 0159

Exception occurs if component model directory is invalid.

EXCEPTION MESSAGES
Given ModelDirectory is invalid.
ModelDirectory "{arg_name}" is invalid.
ModelDirectory "{arg_name}" is invalid. Reason: {reason}.
ModelDirectory "{arg_name}" is invalid. Reason: {reason}. {troubleshoot_hint}

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 component 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: {exception}.
Unknown library exception: {exception}. {customer_support_guidance}.

Execute Python Script component

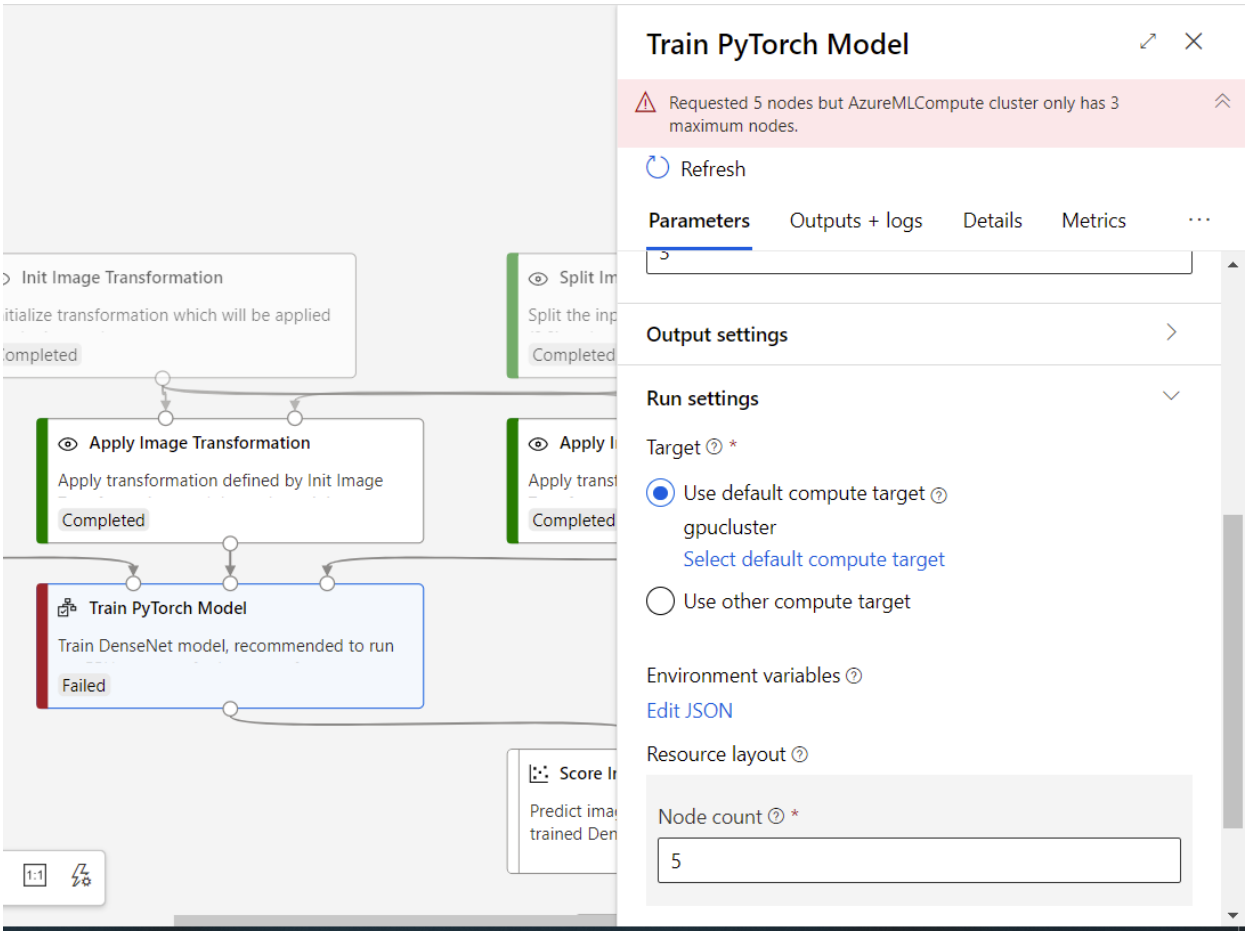
Search **in azureml_main** in **70_driver_logs** of **Execute Python Script component** and you could find which line occurred error. For example, "File "/tmp/tmp01_ID/user_script.py", line 17, in azureml_main" indicates that the error occurred in the 17 line of your Python script.

Distributed training

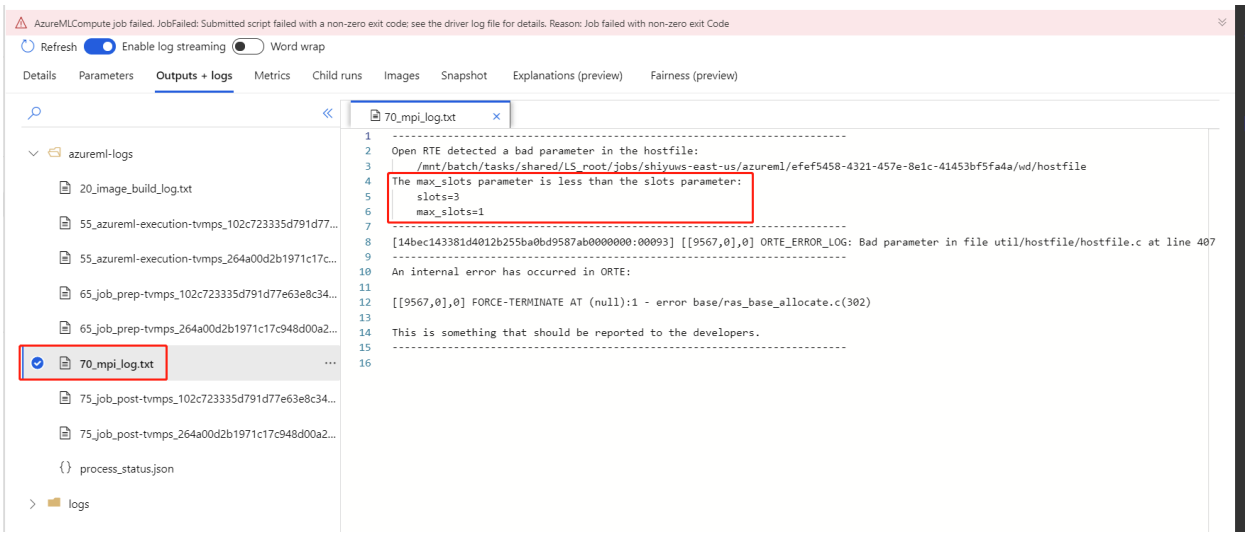
Currently designer supports distributed training for and Train PyTorch Model component.

If the component enabled distributed training fails without any `70_driver` logs, you can check `70_mpi_log` for error details.

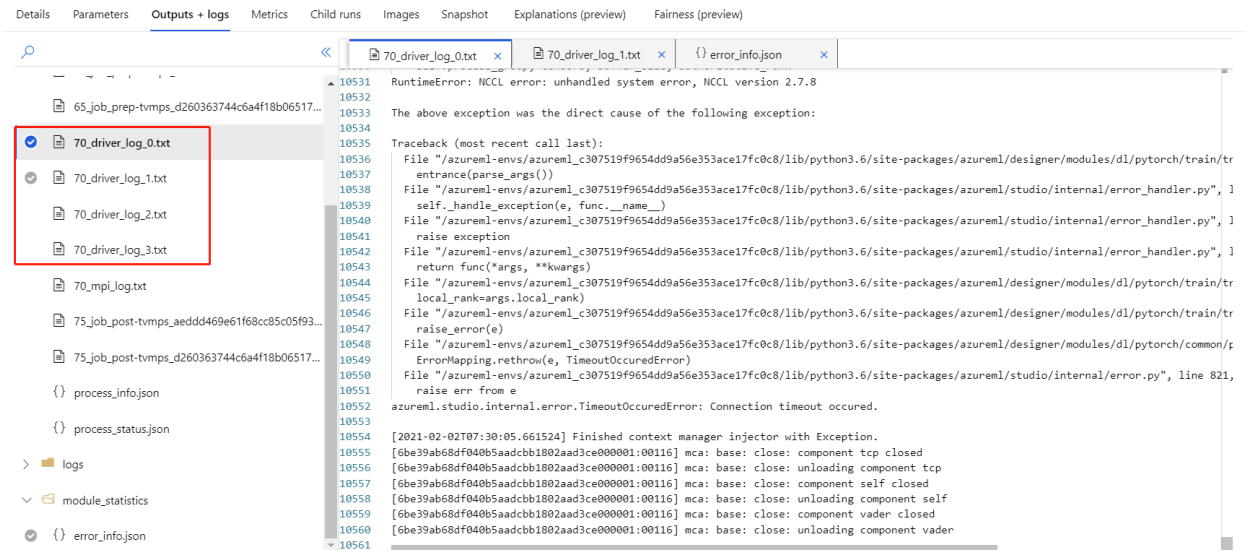
The following example shows that the **Node count** of run settings is larger than available node count of compute cluster.



The following example shows that **Process count per node** is larger than **Processing Unit** of the compute.



Otherwise, you can check `70_driver_log` for each process. `70_driver_log_0` is for master process.

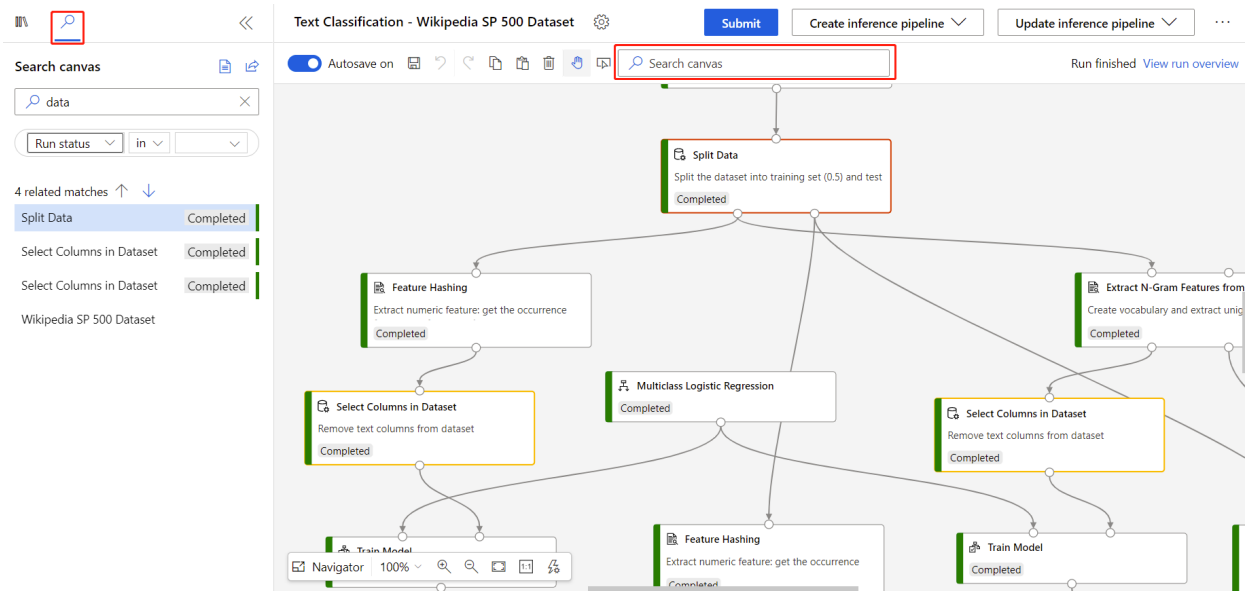


Graph search query syntax

3/28/2022 • 2 minutes to read • Edit Online

In this article, you learn about the graph search functionality in Azure Machine Learning.

Graph search lets you fast navigate a node when you are debugging or building a pipeline. You can either type the key word or query in the input box in the toolbar, or under search tab in the left panel to trigger search. All matched result will be highlighted in yellow in canvas, and if you select a result in the left panel, the node in canvas will be highlighted in red.



Graph search supports full-text keyword search on node name and comments. You can also filter on node property like runStatus, duration, computeTarget. The keyword search is based on Lucene query. A complete search query looks like this:

[[lucene query] | [filter query]]

You can use either Lucene query or filter query. To use both, use the **|** separator. The syntax of the filter query is more strict than Lucene query. So if customer input can be parsed as both, the filter query will be applied.

For example, `data OR model | compute in {cpucluster}`, this is to search nodes of which name or comment contains `data` or `model`, and compute is cpucluster.

Lucene query

Graph search uses Lucene simple query as full-text search syntax on node "name" and "comment". The following Lucene operators are supported:

- AND/OR
- Wildcard matching with **?** and ***** operators.

Examples

- Simple search: `JSON Data`
- AND/OR: `JSON AND Validation`
- Multiple AND/OR: `(JSON AND Validation) OR (TSV AND Training)`

- Wildcard matching:
  - `machi?e learning`
  - `mach*ing`

> **NOTE**
> You cannot start a Lucene query with a "*" character.

Filter query

Filter queries use the following pattern:

`**[key1] [operator1] [value1]; [key2] [operator1] [value2];**`

You can use the following node properties as keys:

- runStatus
- compute
- duration
- reuse
- publish
- tags

And use the following operators:

- Greater or equal: `>=`
- Less or equal: `<=`
- Greater: `>`
- Less: `<`
- Equal: `==`
- Contain: `=`
- NotEqual: `!=`
- In: `in`

Example

- duration > 100;
- status in { Failed,NotStarted}
- compute in {gpu-cluster}; runStatus in {Completed}

Technical notes

- The relationship between multiple filters is "AND"
- If `>=, >, <, or <=` is chosen, values will automatically be converted to number type. Otherwise, string types are used for comparison.
- For all string type values, case is insensitive in comparison.
- Operator "In" expects a collection as value, collection syntax is `{name1, name2, name3}`
- Space will be ignored between keywords