Big Data with Azure Machine Learning

Lab 2 – Building Predictive Models

Overview

In this lab, you will learn how to train and evaluate machine learning models using Azure Machine Learning.

What You'll Need

To complete this lab, you will need the following:

- A Microsoft account (for example, an *outlook.com*, *live.com*, or *hotmail.com* address)
- A Microsoft Azure subscription
- A Windows, Linux, or Mac OS X computer
- The lab files for this course

**Note**: Before starting this lab, you must complete the previous lab in this course.

Exercise 1: Training a Regression Model

In this exercise, you will use Azure Machine Learning to create a regression model. The goal of this model is to predict how many calories a person has expended during an exercise session.

Prepare the Data

As with most machine learning projects, you will start by preparing the data.

1. If you are not already signed into Azure Machine Learning Studio, browse to https://studio.azureml.net and sign in using the Microsoft account associated with your Azure subscription.
2. Open the **Exercise Analysis** experiment you created in the previous lab, which should look like this:
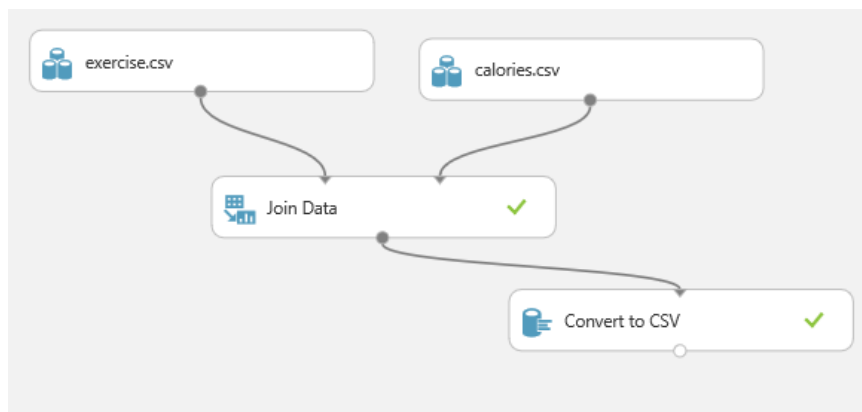
3. Add an **Edit Metadata** module to the experiment and connect the output of the **Join Data** module to the input of the **Edit Metadata** module, as shown here:
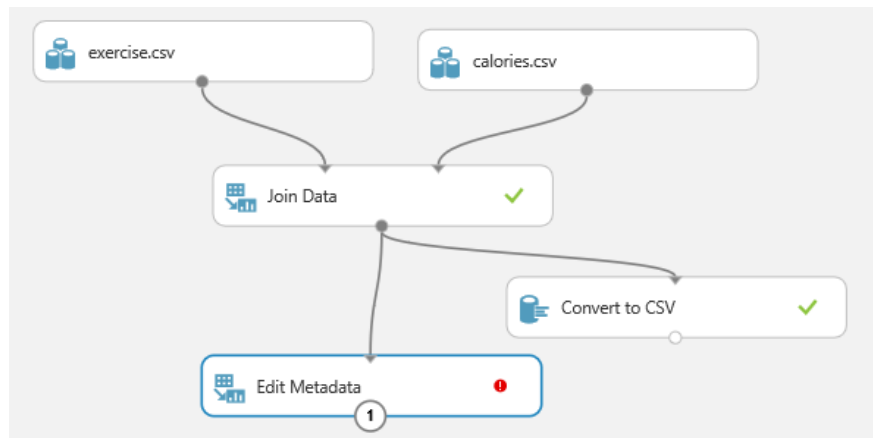
4. Configure the properties of the **Edit Metadata** module as follows:
   - **Column**: Gender
   - **Data type**: Unchanged
   - **Categorical**: Make categorical
   - **Fields**: Unchanged
   - **New column names**: *Leave blank*

   Note: Gender is a feature that is used to categorize people, it is not a measurable value (one person can't have *more* or *less* gender than another!). Configuring this explicitly will help Azure Machine Learning use the feature appropriately when we train a predictive model.

5. Add an **Apply Math Operation** module and connect the output of the **Edit Metadata** module to the input of the **Apply Math Operation** module.

   Note: In most machine learning scenarios, you will need to modify existing features or add new ones based on mathematical transformations to improve the relationships between features and the label you are trying to predict. This process is referred to by data scientists as *feature engineering*. In this case, you will convert **Duration** and **Heart Rate** to second order polynomial features (in other words, the training data will include the value and the value squared). This transformation will help account for some non-linearity in the relationship between these features and the label (**Calories**).

6. Configure the **Apply Math Operation** module as follows:
   - **Category**: Basic
   - **Basic math function**: Pow
   - **Second argument type**: Constant
   - **Constant second argument**: 2
   - **Column set**: *Duration* and *Heart_Rate* (you may need to type these in the **With Rules** tab of the column selector if you have not run the experiment up to this point)
   - **Output mode:** Append
7. Connect the output of the **Edit Metadata** module to the input of the **Apply Math Operation**
8. Select the **Apply Math Operation** module, and run the selected modules. This will run the **Edit Metadata** and **Apply Math Operation** modules you added since the experiment was last run.

9. Visualize the output of the **Apply Math Operation** and select the **Gender** column heading to verify that its **Feature Type** is now *Categorical Feature*; and observe that two new columns, named **Pow(Duration_$2)** and **Pow(Heart_Rate$2)**, have been added to the dataset.
10. Add a second **Apply Math Operation** module to the experiment, and connect it to the output of the existing **Apply Math Operation** module as shown here:
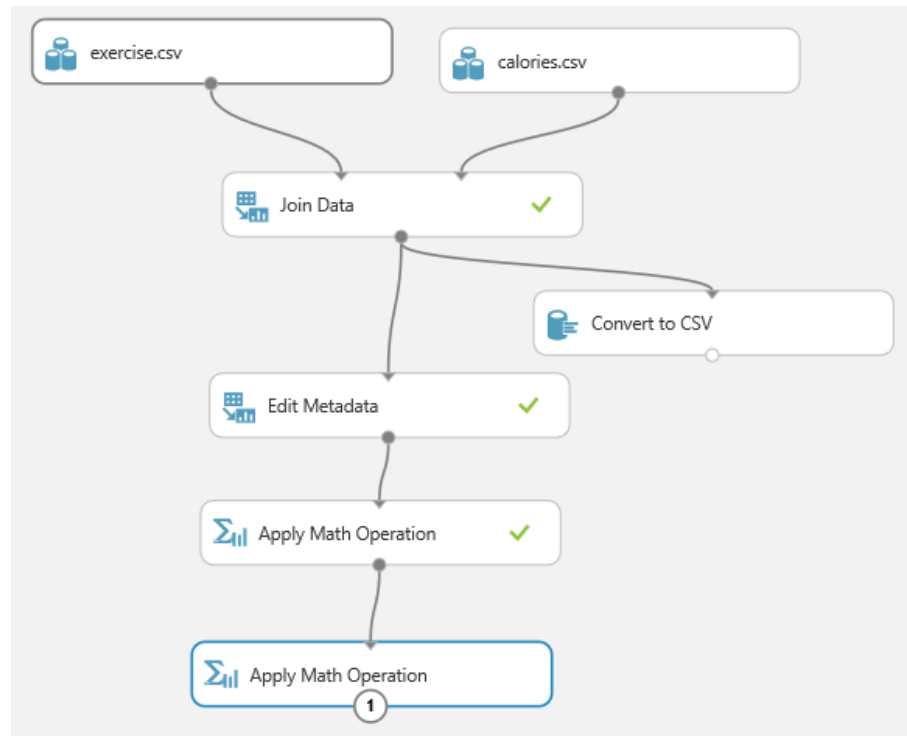
> **Note**: The goal here is to predict the calories burned during an exercise session. Calories burned are a strictly positive quantity – there can be no negative calories. Therefore, you will natural log transform the label, **Calories**, to ensure the regression model does not predict any non-positive values.

11. Configure the new **Apply Math Operation** module as follows:
    - **Category**: Basic
    - **Basic math function**: Ln
    - **Column set**: Calories
    - **Output mode**: Append
12. Select the **Apply Math Operation** module and run the selected module. Then visualize the output of the second **Apply Math Operation** and observe that a new column named **Ln(Calories)** has been added to the dataset.
13. Add another **Edit Metadata** module to the experiment and connect the output of the second **Apply Math Operation** module to its input.

> **Note**: The dataset includes a **User_ID** column and the untransformed **Calories** column. Neither of these columns should be used as a feature when training the model (**User_ID** because it's simply a unique identifier for each user, and any relationship it has with calories burned is coincidental; and **Calories** because that's the original label that we're trying to predict.)

14. To prevent the **User_ID** and the **Calories** columns from confounding model training, configure the **Edit Metadata** module as follows:
    - **Column**: *User_ID* and *Calories*
    - **Data type**: Unchanged
    - **Categorical**: Unchanged
    - **Fields**: Clear feature
    - **New column names**: *Leave blank*
15. Add a **Normalize Data** module to the experiment, and connect the output of the second **Edit Metadata** module to its input.

    Note: It's common when preparing data for model training to scale, or *normalize*, numeric features so that those with large numeric values do not dominate the training of the machine learning model, simply because the values are large. The specific scaling technique used depends on the statistical distribution of the data values - the **Normalize Data** module supports five normalization methods. In this case, the distributions of the numeric columns are quite different. Some of the **Height**, **Weight**, **Heart_Rate**, **Body_Temp** features (and the engineered features based on them) have an approximately *Normal* distribution (visualized as histogram with a bell-shaped curve). **ZScore** (mean-variance) normalization is appropriate here. The **Age** and **Duration** features have distributions far from Normal, so **MinMax** normalization (forcing all values in a limited range of say {0,1}) is more appropriate for these columns. Therefore, you will use two **Normalize Data** modules to perform the normalization of all numeric features.

16. Configure the **Normalize Data** module as follows:
    - **Transform method**: ZScore
    - **Use 0 for constant columns**: Checked
    - **Columns to transform**: *Height*, *Weight*, *Heart_Rate*, Body_*Temp*, and *Pow(Heart_Rate$2)*
17. Add a second **Normalize Data** module to the experiment, and connect the **Transformed dataset** (left) output of the first **Normalize Data** module to its input. Then configure the new **Normalize Data** module as follows:
    - **Transform method**: MinMax
    - **Use 0 for constant columns**: Checked
    - **Columns to transform**: *Age*, *Duration*, and *Pow(Duration_$2)*
18. Select and run the modules that have not yet been run, and then verify that your experiment looks like this:
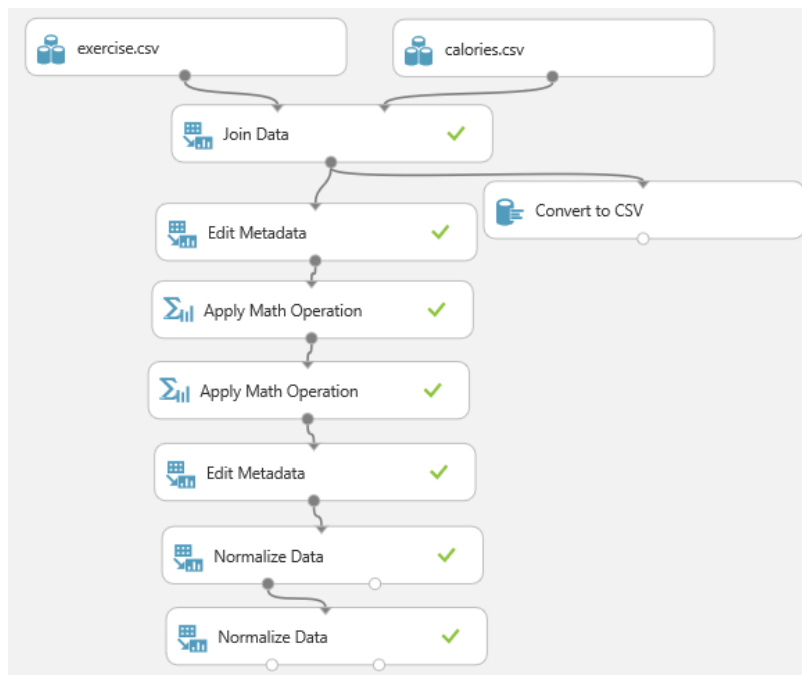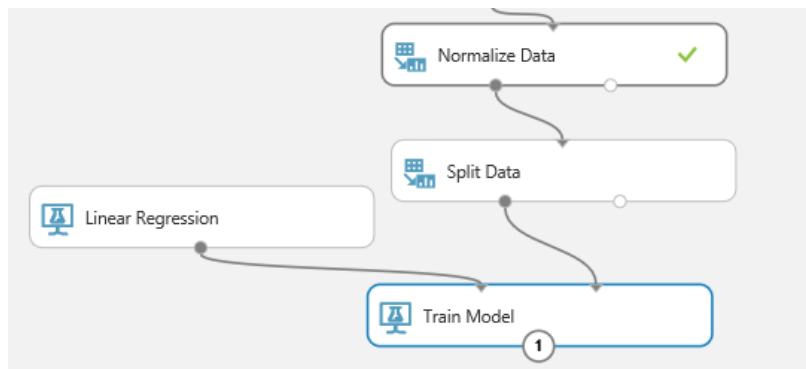
Train a Regression Model

Now that you have prepared the data, you will train a regression machine learning model to predict the calories burned by a person during exercise.

1. Add a **Split Data** module to the experiment, and connect the **Transformed dataset** (left) output of the last **Normalize Data** module to its input.
2. Configure the **Split Data** module as follows:
   - **Splitting mode**: Split Rows
   - **Fraction of rows in the first output dataset**: 0.7
   - **Randomized split**: Checked
   - **Random seed**: 1234
   - **Stratified split**: False

   Note: Regression is a supervised learning technique in which the training data includes labels for the value we are trying to predict. We can therefore split the data into a training set with which to train the model, and a test set to compare how well the model predicts the label compared to the known values.

3. Add a **Train Model** module to the experiment, and connect the **Results dataset1** (left) output of the **Split Data** module to its **Dataset** (right) input.
4. Configure the **Train Model** module properties to set the **Label** column to **Ln(Calories)**. This is the value you will train the model to predict.
5. Add a **Linear Regression** module to the experiment and connect its output to the **Untrained model** (left) input of the **Train Model** module, as shown here:

6. Configure the **Linear Regression** module properties as follows:
   - **Solution method**: Ordinary Least Squares
   - **L2 regularization weight**: 0.1
   - **Included intercept term**: Unchecked
   - **Random number seed**: 1234
   - **Allow unknown categorical levels**: Checked
7. Add a **Score Model** module to the experiment, and connect the output of the **Train Model** module to its **Trained model** (left) input. Then connect the **Result dataset2** (right) output of the **Split Data** module to its **Dataset** (right) input, as shown here:
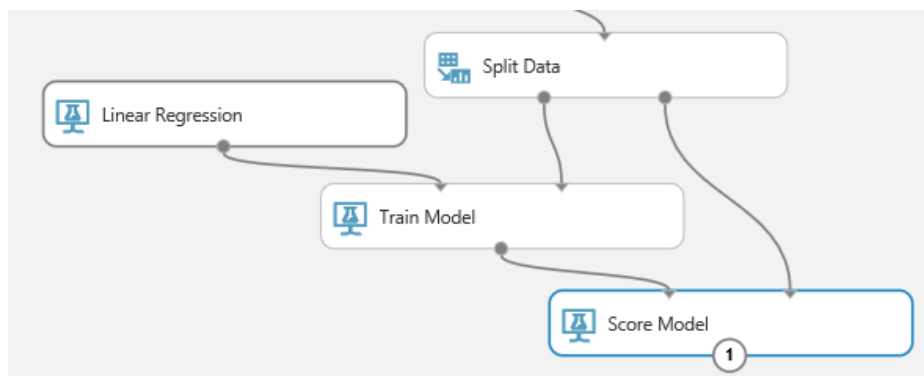

8. Save and run the experiment.
9. When the experiment has finished, visualize the output of the **Train Model** module, which shows the settings used when training the model and the relative weights (or importance) of the features used by the trained model to predict **Ln(Calories)**. Note that **Duration** feature appears to have the largest impact on predicting calories burned.
10. Visualize the output of the **Score Model** module, and compare the values in the **Scored Labels** column (which were predicted by the model) to those in the **Ln(Calories)** column (which are the actual log-normal values for calories.)

Evaluate the Model

1. Add an **Evaluate Model** module to the experiment. and connect the output of the **Score Model** module to its **Scored dataset** (left) input, as shown here:
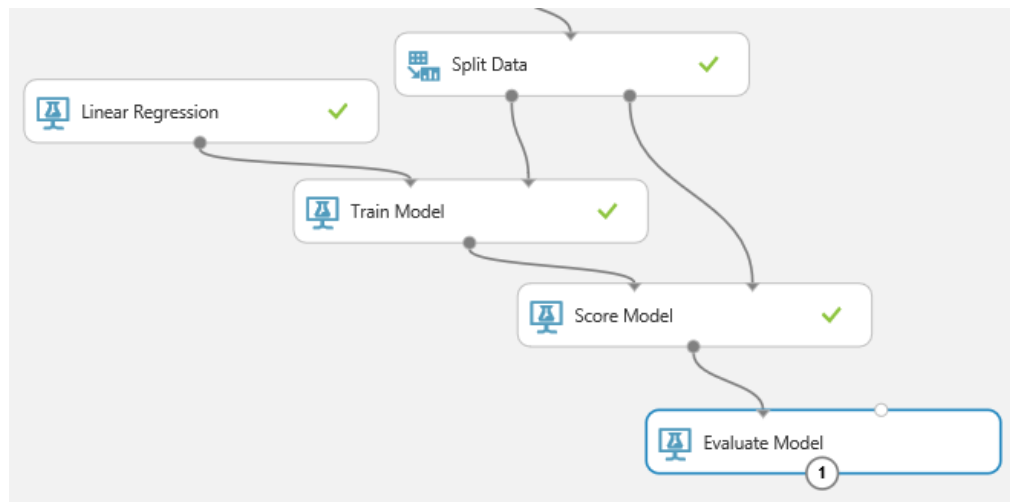
2. Select and run the **Evaluate Model** module.
3. Visualize the output of the **Evaluate Model** module, which should resemble the figure below.
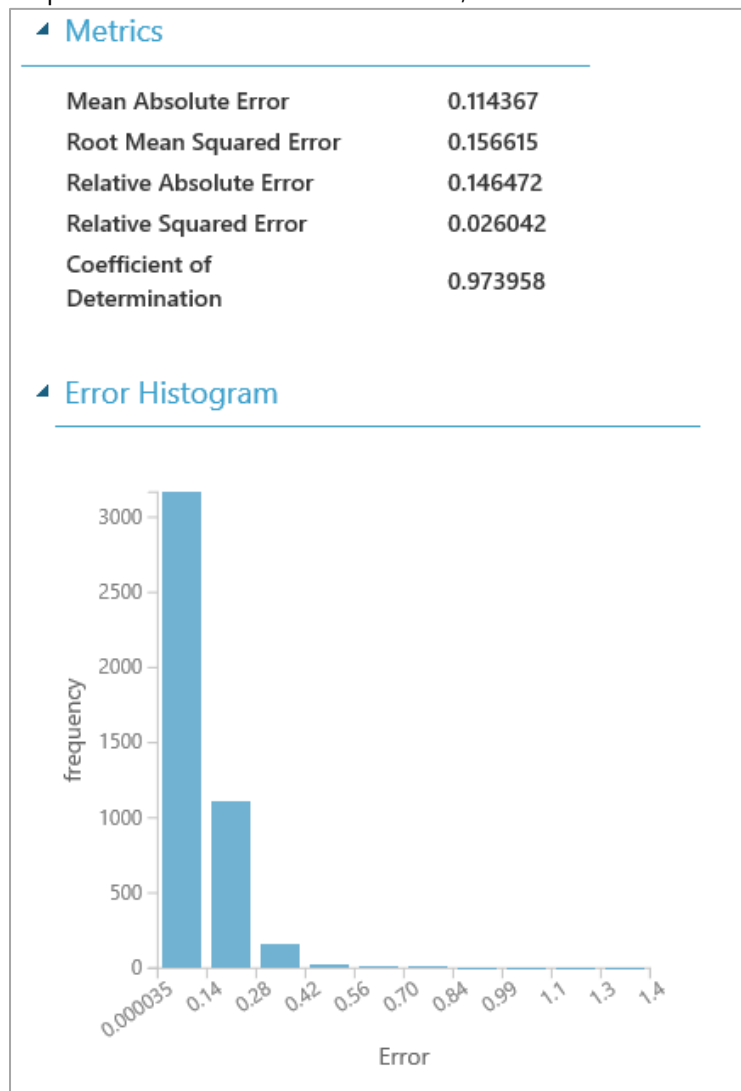

Overall these metrics are quite good, and the model is satisfactory. Specifically, notice the following (keeping in mind that the label is on a natural log scale):

- **Mean Absolute Error**, **Root Mean Squared Error** are both fairly small indicating most errors are small.
- **Relative Absolute Error** and **Relative Squared Error**, which measure error relative to the label value, are also small.
- The **Coefficient of Determination**, which measures the reduction in variance for the model relative to the label values is high. A Coefficient of Determination of 0 indicates the model is not explaining the label, whereas a value of 1.0 indicates perfect correspondence between label and score (and likely indicates overfitting!).
- The histogram of the residuals (absolute difference between the label and the score) shows most the residuals are small and clustered near zero. There are a few outliers, which are label values the model did not predict accurately.

To get an additional feel for the errors in the predictions (scores) from this model you can compare a few label values to the scores in the original units of calories.

4. Add an **Apply Math Operation** module, and connect the output of the **Score Model** module to its input.
5. Configure the **Apply Math Operation** module as follows:
   - **Category**: Basic
   - **Basic math function**: Exp
   - **Column set**: Scored Labels
   - **Output mode**: Append
6. Run the **Apply Math Operation** module.
7. Visualize the output of the **Apply Math Operation** module, and compare the values in the of the **Calories** column with the **Exp(Scored Labels)** column as shown below.

Calories	Pow (Duration_$2)	Pow (Heart_Rate_$2)	Ln (Calories)	Scored Labels	Exp(Scored Labels)
172	0.870968	0.873163	5.147494	4.999393	148.323164
99	0.359288	-0.207123	4.59512	4.805363	122.163881
178	0.694105	2.187511	5.181784	5.115123	166.521326
55	0.070078	0.427955	4.007333	3.941849	51.513759
187	0.750834	0.987193	5.231109	5.090221	162.425705
27	0.038932	-0.706346	3.295837	3.256751	25.96505
125	0.359288	0.427955	4.828314	4.816893	123.58059
18	0.038932	-1.086082	2.890372	2.94997	19.105372
25	0.088988	-0.992785	3.218876	3.28301	26.655886

Exercise 2: Training a Classification Model

In this exercise, you will use Azure Machine Learning to create a classification model. The goal of this model is to predict whether a patient is diabetic based on diagnostic measurements.

As is often the case with machine learning of any kind, a fair amount of data preparation is required before you can use the data to train a model.

1. In Azure Machine Learning Studio, open the **Pima Big Data** experiment you created in Lab 1, and verify that it looks like this:

2. Add an **Apply Math Operation** module to the experiment, and connect the output of the **Join Data** module to its input.
3. Configure the **Apply Math Operation** module properties as follows:
    - **Category**: Basic
    - **Basic math function**: Ln
    - **Column set**: Age
    - **Output mode**: Append
4. **Save** your experiment and **Run selected** to update the output schema of the **Apply Math Operation** module to verify that the **Ln(Age)** column has been added to the dataset.
5. Add a **Normalize Data** module to the experiment and connect the output of the **Apply Math Operation** module to its input.
6. Configure the **Normalize Data** module properties as follows:
    - **Transformation method**: ZScore
    - **Use 0 for constant columns when checked**: Checked
    - **Select columns**:
        PlasmaGlucose
        DiastolicBloodPressure
        TriceptsThickness
        SerumInsuin
        BMI
7. Add a second **Normalize Data** module to the experiment, and connect the **Transformed dataset** (left) output of the first **Normalize Data** module to its input.
8. Configure the new **Normalize Data** module as follows
    - **Transformation method**: MinMax
    - **Use 0 for constant columns when checked**: Checked
    - **Select columns**:
        Pregnancies
        DiabetesPedigree
        Age
        Ln(Age)
9. Save your experiment and select and run the **Normalize Data** modules.
10. Add an **Edit Metadata** module to the experiment and connect the **Transformed dataset** (left) output of the second **Normalize Data** module to its input.
11. Configure the **Edit Metadata** module as follows:

- **Selected columns**: *PatientID* and *Physician*
- **Data type**: Unchanged
- **Categorical**: Unchanged
- **Fields**: Clear feature
- **New column names**: *Leave blank*

12. Save and run the experiment.
13. Verify that the experiment looks like this.
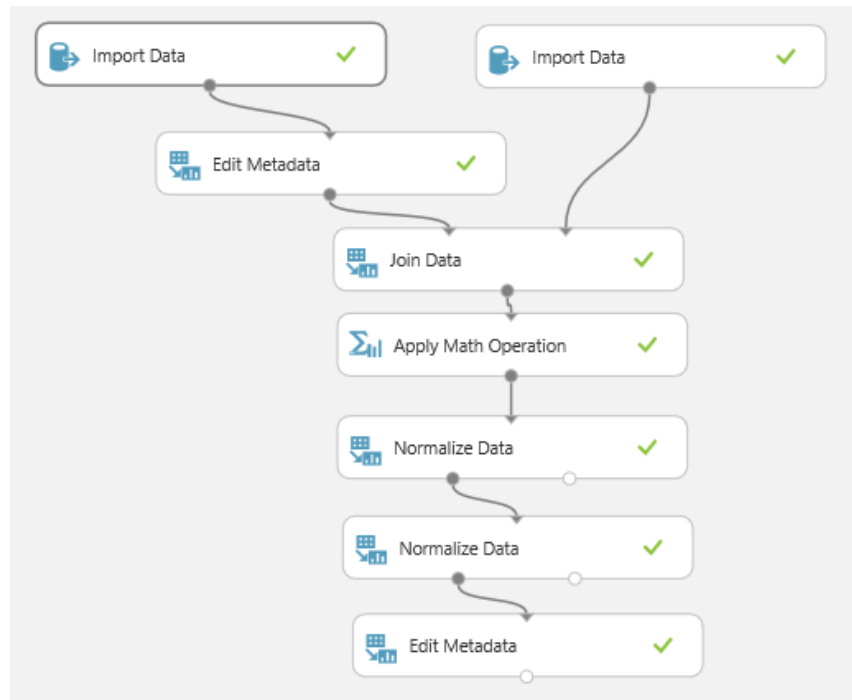
Train and Evaluate the Classification Model

Now that you have prepared the data set, you will use these data to train and evaluate a classifier machine learning model.

1. Add a **Split Data** module to the experiment and connect the output of the **Edit Metadata** module to its input.
2. Configure the **Split Data** module as follows:
   - **Splitting mode**: Split Rows
   - **Fraction of rows in the first output dataset**: 0.7
   - **Random seed**: 1234
   - **Stratified split**: False
3. Add a **Train Model** module to the experiment and connect the **Results dataset1** (left) output of the **Split Data** module to its **Dataset** (right) input.
4. Configure the **Train Model** module properties to set the **Label column** to **Diabetic**.
5. Add a **Two Class Logistic Regression** module to the experiment, and connect its output to the **Untrained model** (left) input of the **Train Model** module.
6. Configure the **Two Class Logistic Regression** module as follows:
   - **Create trainer mode**: Single Parameter
   - **Optimization tolerance**: 1E-07
   - **L1 regularization weight**: 1
   - **L2 regularization weight**: 1
   - **Memory size for L-BFGS**: 20

- **Random number seed**: 1234
- **Allow unknow categories**: Checked

7. Add a **Score Model** module to the experiment, and connect the output of the **Train Model** module to its **Trained model** (left) input. Then connect the **Result dataset2** (right) output of the **Split Data** module to its **Dataset** (right) input.

8. Add an **Evaluate Model** module to the experiment, and connect the output of the **Score Model** module to its **Scored dataset** (left) input.

9. Verify that the lower portion of your experiment resembles the figure below.
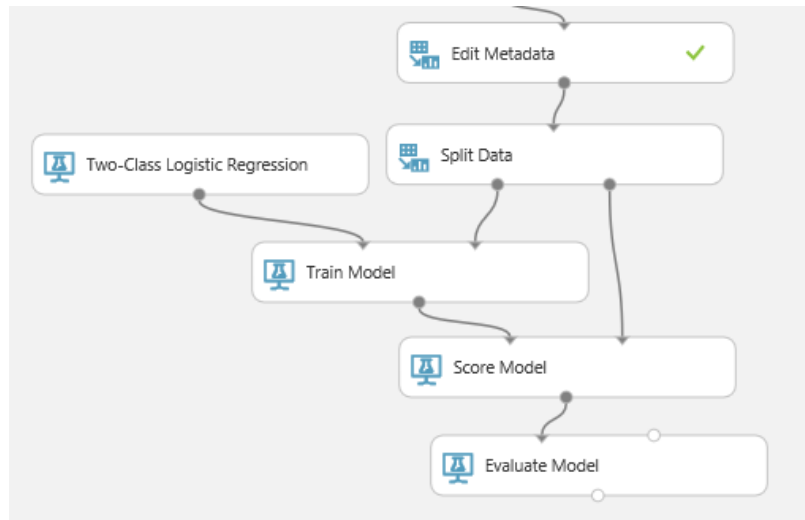
10. Save and run the experiment.
11. Visualize the output of the **Score Model** module, and examine the results

    Compare the label column (**Diabetic**), which indicates is the patient is actually diabetic or not, and the **Scored Labels** column which contains the model predictions for diabetic patients. Notice that most of the cases are classified correctly, but that there are some errors.

12. To examine summary statistics of the visualize the output of the **Evaluate Model** module and examine the results. View the ROC curve, which should resemble the figure below.
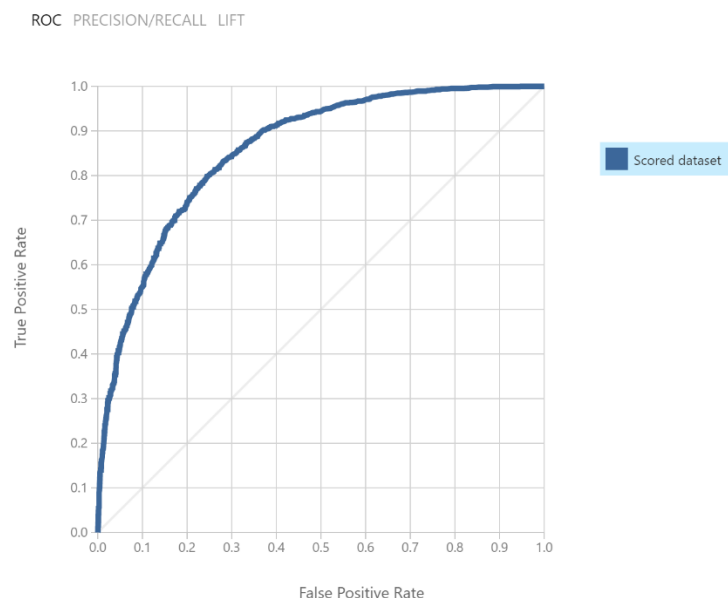
The ROC curve shows the trade-off between the True Positive Rate (positive cases correctly classified) and False Positive Rate (positive cases incorrectly classified). This curve is above and to the left of the light 45 degree line. This indicates that the classifier is more effective than simply guessing.

13. Scroll down and examine the performance metrics, which should appear as shown below.

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
904	603	0.787	0.717	0.5	0.858

False Positive	True Negative	Recall	F1 Score		
357	2636	0.600	0.653		

Positive Label	Negative Label
1	0

Notice the following about these metrics:
- The **Confusion Matrix** shows the number of True Positives and True Negatives (cases correctly classified) and False Negatives and False Positives (cases incorrectly classified).
- The **AUC** (Area Under the Curve) is the area under the ROC curve. A perfect classifier would have an AUC of 1.0, indicating no trade-off between True and False Positive Rates.
- **Accuracy** is the fraction of cases correctly classified.
- **Recall**, is the fraction of positive cases correctly classified. Notice this figure is only 0.585, which means the classifier misclassifies more than 4 of 10 diabetic patients as non-diabetic.
- **Precision** is the fraction of negative cases correctly classified.

Overall, this classifier is significantly better than random guessing, but has only limited accuracy.

Try a Different Algorithm

Given the questionable performance of the classification model, a data scientist would normally work on ways to improve the result. One approach is to find an alternative model algorithm that fits the particular problem better.

1. Add a second **Train Model** module to the experiment and connect the **Results dataset1** (left) output of the **Split Data** module to its **Dataset** (right) input (in addition to the existing connection to the original **Train Model** module.)
2. Configure the properties for the new **Train Model** to set the **Label column** to **Diabetic**.
3. Add a **Two Class Boosted Decision Tree** module to the experiment, and connect its output to the **Untrained model** (left hand) input of the new **Train Model** module.
4. Configure the **Two Class Boosted Decision Tree** module properties as follows:
   - **Create trainer mode**: Single Parameter
   - **Maximum number of leaves per tree**: 20
   - **Minimum number of training instances required to form a leaf**: 10
   - **Learning rate**: 0.2
   - **Number of trees constructed**: 100
   - **Random number seed**: 1234
   - **Allow unknown categories**: Checked
5. Add a second **Score Model** module to the experiment, and connect the output of the **Train Model** module for the **Two Class Boosted Decision Tree** module to its **Trained model** (left) input. Then connect the **Result dataset2** (right) output of the **Split Data** module to its **Dataset** (right) input. A shown here.

6. Connect the output of the new **Score Model** module to the **Scored dataset to compare** (right) input of the **Evaluate Model** module, as shown here:
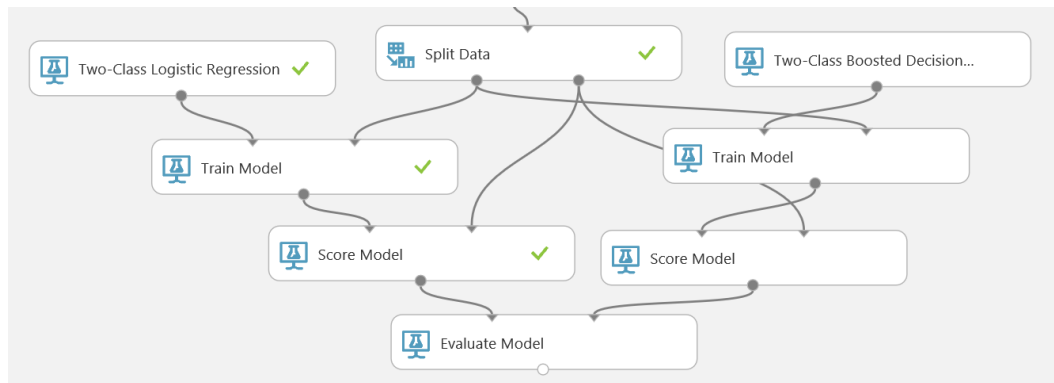
7. Save and run the experiment.
8. Visualize the output of the **Evaluate Model** module, and in the legend for the ROC chart, select **Scored dataset to compare** (which reflects the output from the **Two-Class Boosted Tree** model). Note that the curve for this model is significantly higher and more to the left than the curve for the **Two-Class Logistic Regression** model:
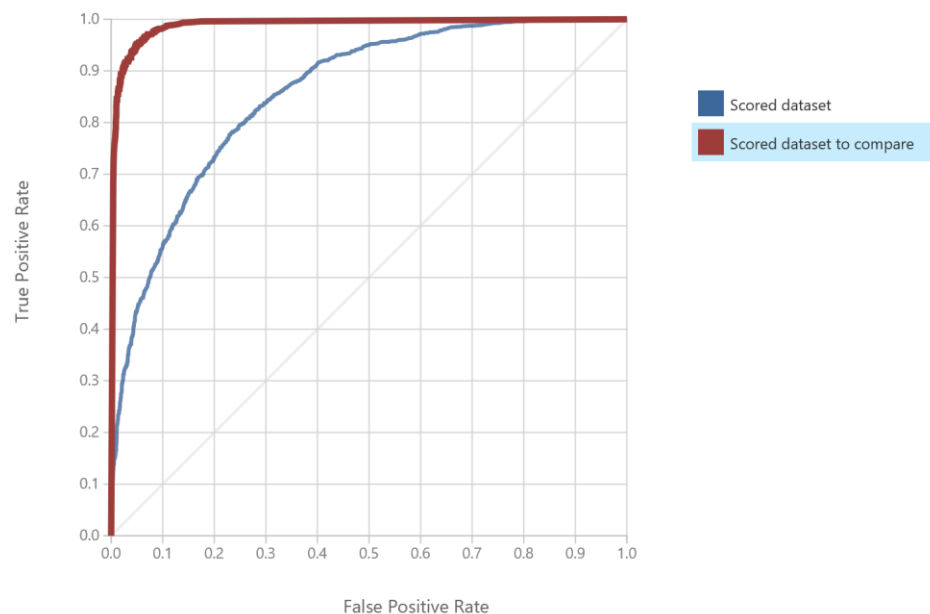
9. With **Scored dataset to compare** selected, scroll down to the performance metrics which should resemble the figure below.

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
1396	111	0.950	0.925	0.5	0.990

False Positive	True Negative	Recall	F1 Score		
113	2880	0.926	0.926		

Positive Label	Negative Label				
1	0				

Compare these results to those achieved with the Two Class Logistic Regression, noticing the following:

- The **AUC** is nearly 0.99, indicating the classifier is nearly ideal.
- The **Accuracy** is now 95%.
- **Recall** is now nearly 0.926.
- **False Positives** have been greatly reduced, reflected by the **Precision** of 0.925.

This classifier is much more satisfactory when compared to the previous one.

10. Delete the **Two-Class Logistic Regression** module and its **Train Model** and **Score Model** modules, and then switch the output connection from the **Scored Model** module for the **Two-Class Boosted Tree** model to the **Scored dataset** (left) input of the **Evaluate Model** module as shown here.
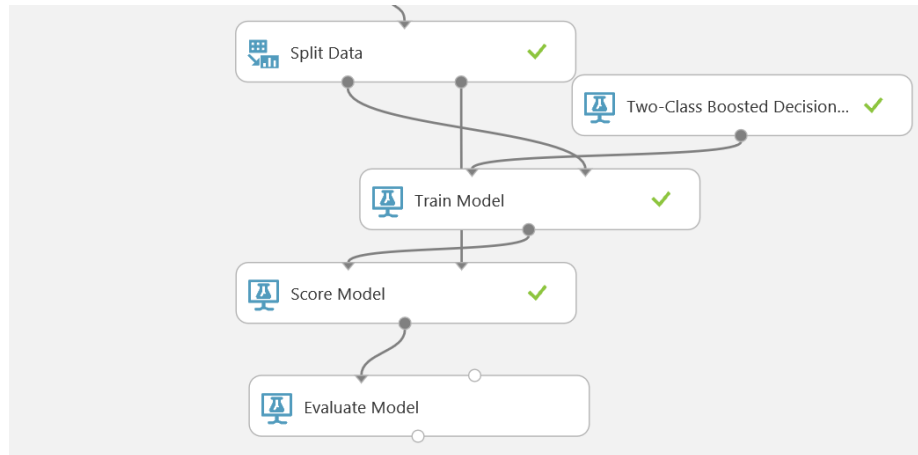
11. Save and run the experiment, and visualize the output of the **Evaluate Model** module to verify that only the **Two-Class Boosted Tree** model is now reflected in the results.

Summary

In this lab, you have used Azure Machine Learning to train and evaluate a regression model and a classification model. This required you to prepare the data for modeling through feature engineering and normalization, splitting the data into training and test datasets, and evaluating performance metrics for regression and classification.

Now you're ready to learn how to productionize your machine learning models as predictive web services.