CIS5560 Term Project Tutorial

Authors: Maria Boldina, Hai Anh Le, Neha Gupta,

Instructor: Jongwook Woo

Date: 05/15/2018

Lab Tutorial

Maria Boldina mboldin@calstatela.edu

Hai Anh Le hle55@calstatela.edu

Neha Gupta ngupta8@calstatela.edu

Predicting Ad Click Fraud using Azure Machine Learning

Objectives

In this tutorial you will learn how to predict whether an app will be downloaded or not after clicking on an ad by training and evaluating two different two-class classification algorithms.

Classification is a machine learning method that uses data to determine the category, type, or class of an item or row of data. Classification is a supervised machine learning method which means that it always requires a labeled training dataset. Classification tasks are frequently organized by whether a classification is binary (either 1 or 0) or multiclass (multiple categories that can be predicted by using a single model). In our case we will be using the binary (two-class) classification to predict whether an app was downloaded (1) or not downloaded (0).

In this hands-on lab, you will learn how to use Azure ML studio to:

1. Load and Prepare data (Partition and Sample, Select Columns, etc.)
2. Learn how to deal with imbalanced datasets (SMOTE & Stratified Split)
3. Create and train the following classification models:
    - Two-Class Decision Jungle
    - Two-Class Decision Forest
4. Improve the model by pruning features and sweeping parameter settings (Tune Model Hyperparameters, Permutation Feature Importance, Cross Validation, etc.)
5. Evaluate the model based on the best metrics
6. Display visualizations

Platform Spec

- Microsoft Azure Machine Learning Studio
- Free Workspace
- Number of nodes: 1
- Storage Size: 10 GB
- Number of modules per experiment: 100
- Region: South Central US

What You'll Need

To complete this lab, you will need the following:

- An Azure ML account
- A web browser and Internet connection
- The dataset https://drive.google.com/file/d/1UHEOMbgsIjl-c2LOUghI9g4g3wMC2IhU/

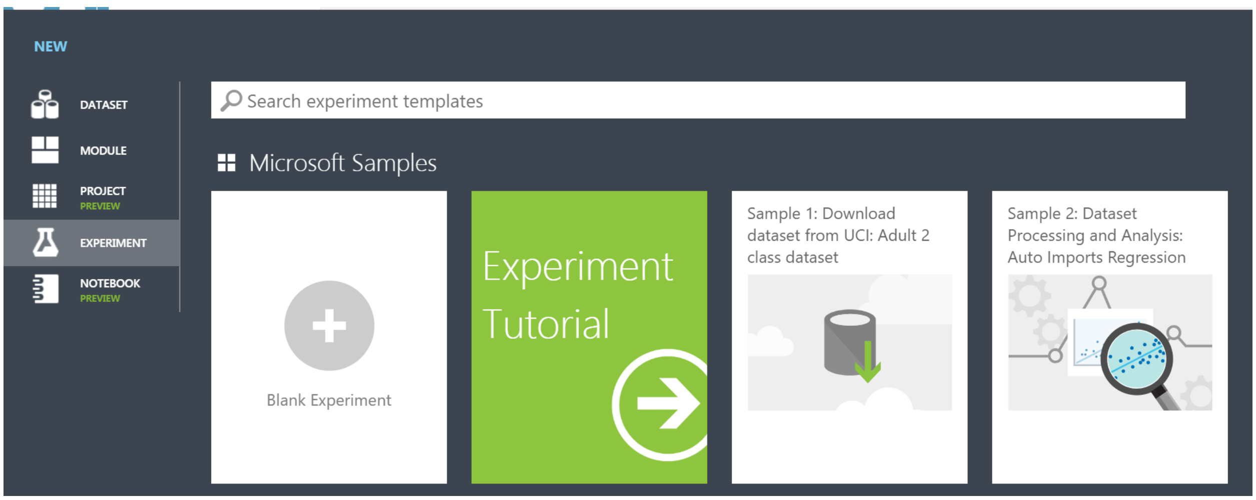
Load and Prepare Data

Step 1: Upload the Dataset

The original dataset from Kaggle contains 200 million clicks over 4-day period provided by TalkingData. TalkingData is China's largest independent big data service platform which covers over 70% of active mobile devices nationwide and handles 3 billion clicks per day, 90% of which are potentially fraudulent.

The goal of our project is to predict whether a user will download an app after clicking on a mobile app ad in order to better target the ads to the audience, to avoid fraudulent practices and save money. Since the original dataset is over 7GB in size we have implemented a Python code to reduce the size to 1GB in order to be able to run the models in Azure ML.

1. Open a browser and browse to https://studio.azureml.net and sign in using the Microsoft account associated with your **Azure ML account** or create an account for Azure Machine Learning Studio if you have not done so before.
2. Download the 1GB dataset from https://drive.google.com/open?id=1UHEOMbgsIjl-c2LOUghI9g4g3wMC2IhU to your computer.
3. In Azure ML on the left-hand side click on EXPERIMENT, then create a new Blank Experiment and give it a name **Two Class Decision Jungle.**

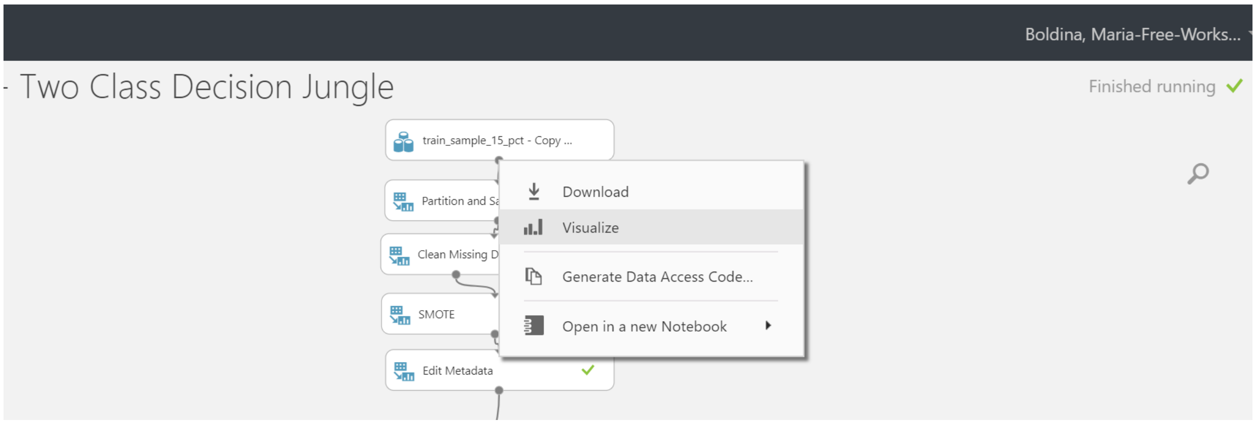


4. **Upload the dataset** to Azure ML by clicking **+NEW** at the bottom of the Machine Learning Studio window, select DATASET tab on the left, and then click on From Local File.
5. In the **Upload a new dataset** dialog box, browse to select the **train_sample_15_pct.csv** file from the folder where you downloaded the dataset file and enter the following details, and then click **OK** button:

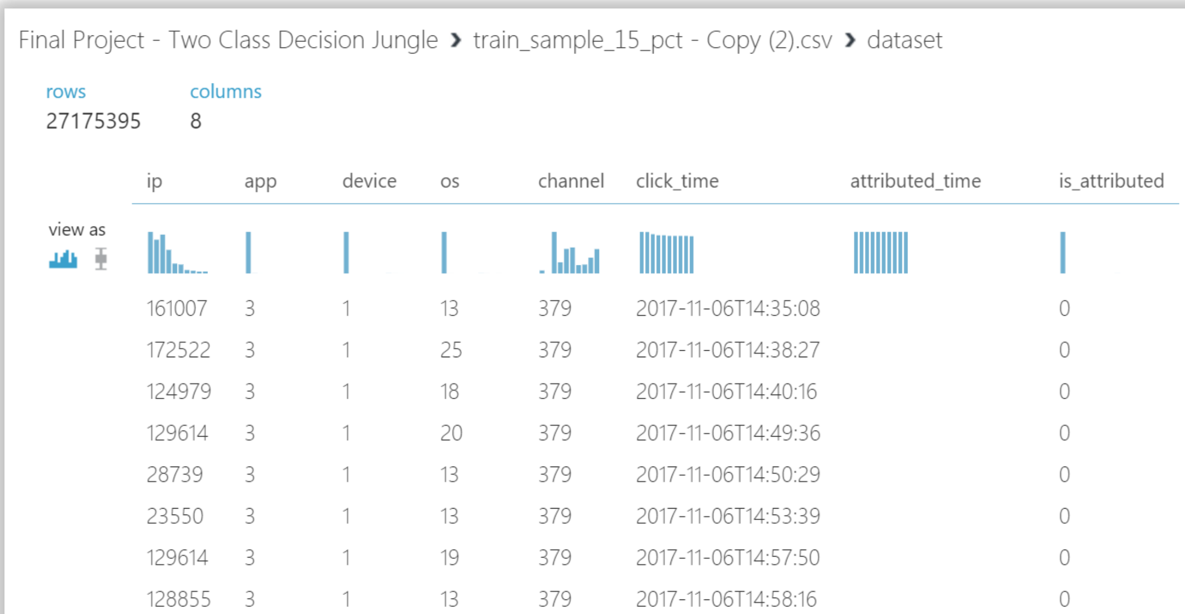
   - **This is a new version of an existing dataset**: Unselected
   - **Enter a name for the new dataset**: train_sample_15_pct
   - **Select a type for the new dataset**: Generic CSV file with a header (.csv)
   - **Provide an optional description**: predicting click fraud

6. Wait for the upload of the dataset to be completed, and then on the left-hand side items pane click on **DATASETS** to verify that the **train_sample_15_pct** dataset is listed.

7. Open your **Two Class Decision Jungle** EXPERIMENT then on the left-hand side items pane search for the **train_sample_15_pct** dataset and drag it to the EXPERIMENT canvas
8. To view the dataset **Right click** on the dataset output port and select **Visualize** it as shown in the image below:



Here is what our dataset looks like:


Final Project - Two Class Decision Jungle › train_sample_15_pct - Copy (2).csv › dataset

rows 27175395 columns 8

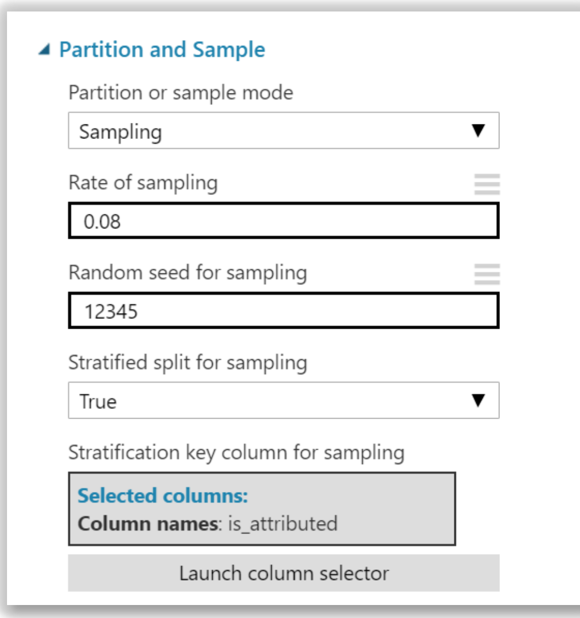
ip	app	device	os	channel	click_time	attributed_time	is_attributed
161007	3	1	13	379	2017-11-06T14:35:08		0
172522	3	1	25	379	2017-11-06T14:38:27		0
124979	3	1	18	379	2017-11-06T14:40:16		0
129614	3	1	20	379	2017-11-06T14:49:36		0
28739	3	1	13	379	2017-11-06T14:50:29		0
23550	3	1	13	379	2017-11-06T14:53:39		0
129614	3	1	19	379	2017-11-06T14:57:50		0
128855	3	1	13	379	2017-11-06T14:58:16		0

In the dataset, we have 8 columns. **Feature columns** include: ip, app, device, OS, channel, click_time, attributed_time. **Label column**: is_attributed (this is the column we are trying to predict).

Step 2: Preprocess and Prepare Data

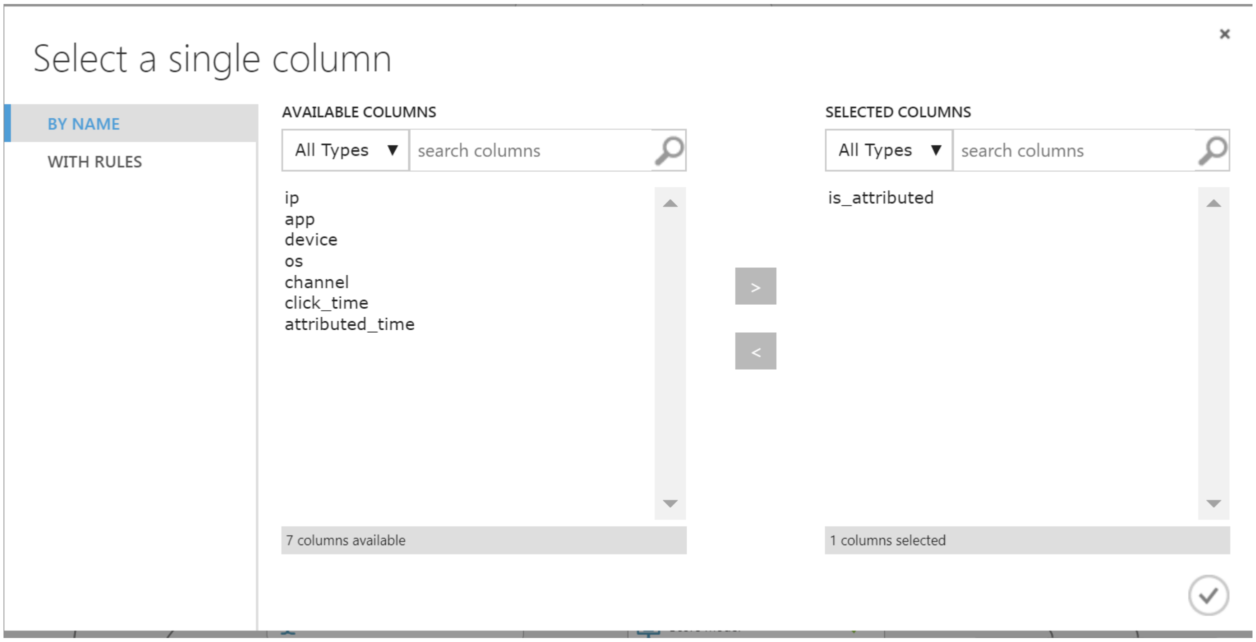
Since our dataset is still too large for Azure ML, we would need to Partition and Sample it to create a smaller dataset to run faster (8 % of the 1GB dataset). In addition, we are dealing with a highly imbalanced dataset where the number of negative class (0) far outweighs the positive class (1). We have only 0.19% of the cases where the app was downloaded vs 99.81% where the app was not downloaded. Therefore, we would have to use SMOTE and Stratified Split techniques to balance the data and ensure that that the smaller output dataset contains the same percentage of the 1s and 0s in the is_attributed column as the original dataset.

1. On the left-hand side items pane search for the **Partition and Sample** module and drag it to the EXPERIMENT canvas below the dataset. Connect the output from the dataset to the Partition and Sample input. Then click on it and set the parameters in the **Properties** pane on the right as indicated in the image below:

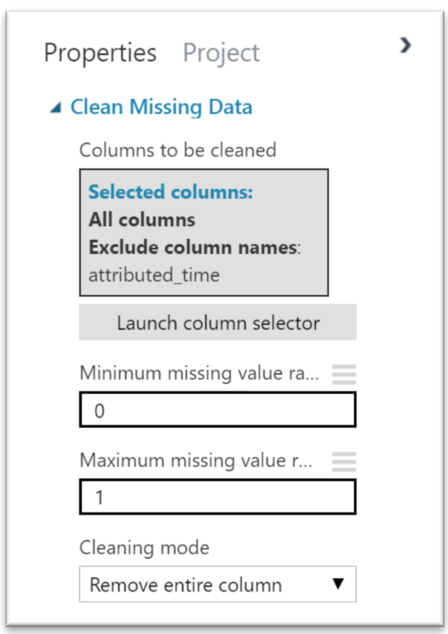


Make sure to set **Stratified Split** to **True** as it ensures that the output dataset contains a representative sample of the values in the selected column.

Click on **Launch Column Selector** and select just the **is_attributed** column as per below



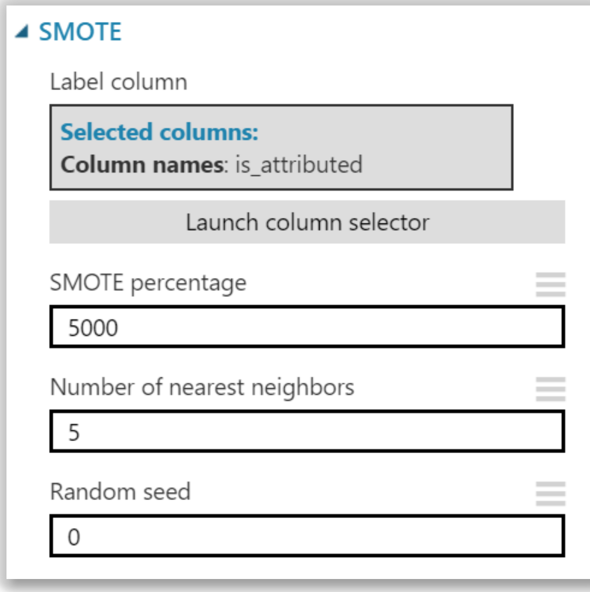
2. On the left-hand side items pane search for the **Clean Missing Data** module and drag it to the EXPERIMENT canvas. Connect the output from Partition and Sample module to the input of Clean Missing Data module. Then click on it and set the parameters in the **Properties** pane on the right as indicated in the image below:



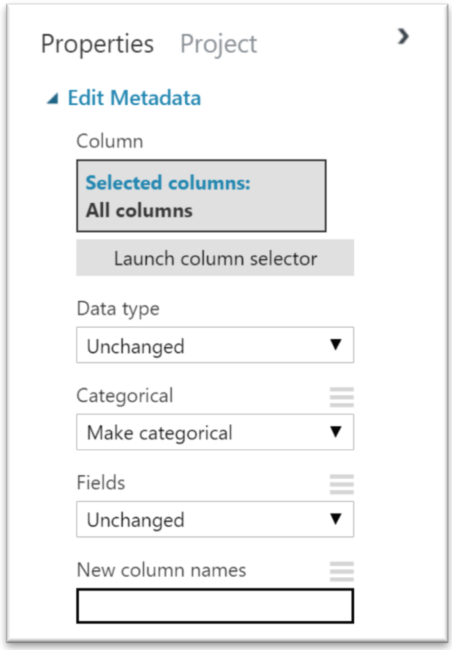
3. **SMOTE:** Synthetic Minority Over Sampling Technique takes a subset of data from the minority class and creates new synthetic similar instances. It helps balance data & avoid overfitting.

Setting the SMOTE percentage to 5000 in our example will increased percent of minority class (1) from 0.19% to 11%.

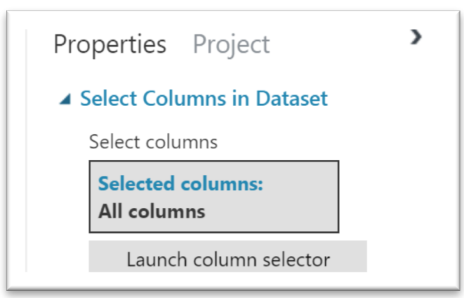
On the left-hand side items pane search for the **SMOTE** module and drag it to the EXPERIMENT canvas. Connect the left output from **Clean Missing Data** to the input of SMOTE module. Then click on it and set the parameters in the **Properties** pane on the right as indicated in the image below:



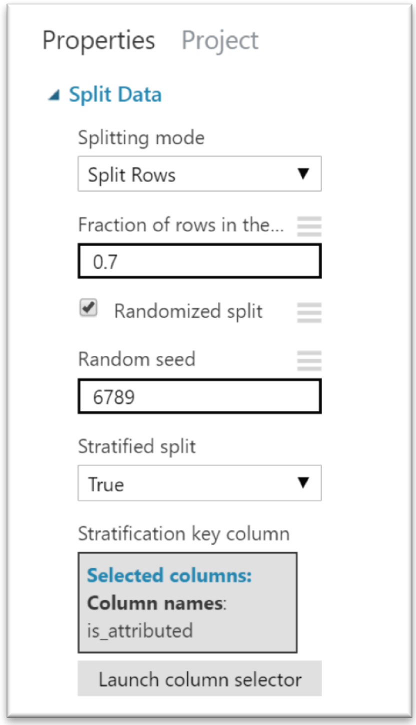
4. Next search for **Edit Metadata,** drag it to the canvas and connect its input to the output of SMOTE. Set the properties as in the image below:



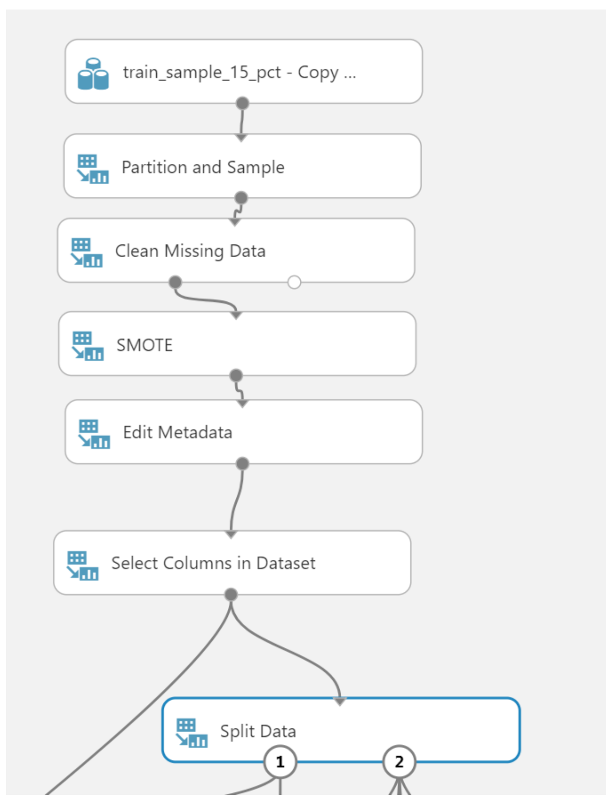
5. Finally, search for **Select Columns in Dataset** module and connect it to the previous module's output. Select all columns in the properties as per below:



6. Search for the **Split Data** module. Drag this module onto your experiment canvas. Connect the **Results dataset** output port of the Select Columns in Dataset module to the **Dataset** input port of the **Split Data** module. Set the Properties of the Split Data module as follows:



Your experiment should now look like the image below:

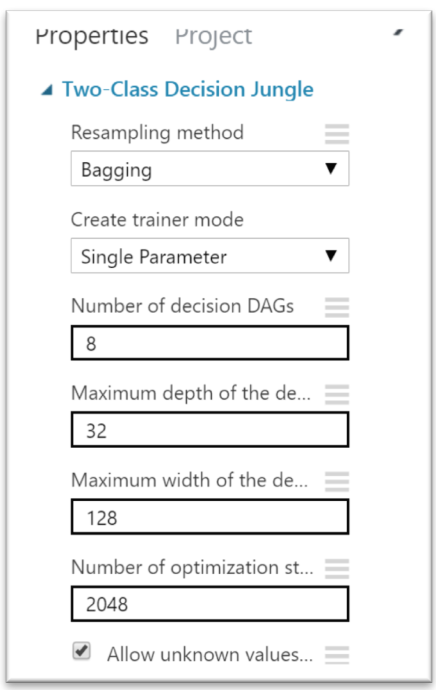


Step 3: Create and Train classification models

Now that the data is ready we can start building our classification models. Given that decision trees often perform well on imbalanced datasets because their hierarchical structure allows them to learn signals from both classes and tree ensembles almost always outperform singular decision trees we have selected the following classification algorithms for our project: Two-class Decision Jungle and Two-class Decision Forest.

Build Model #1: Two-class Decision Jungle

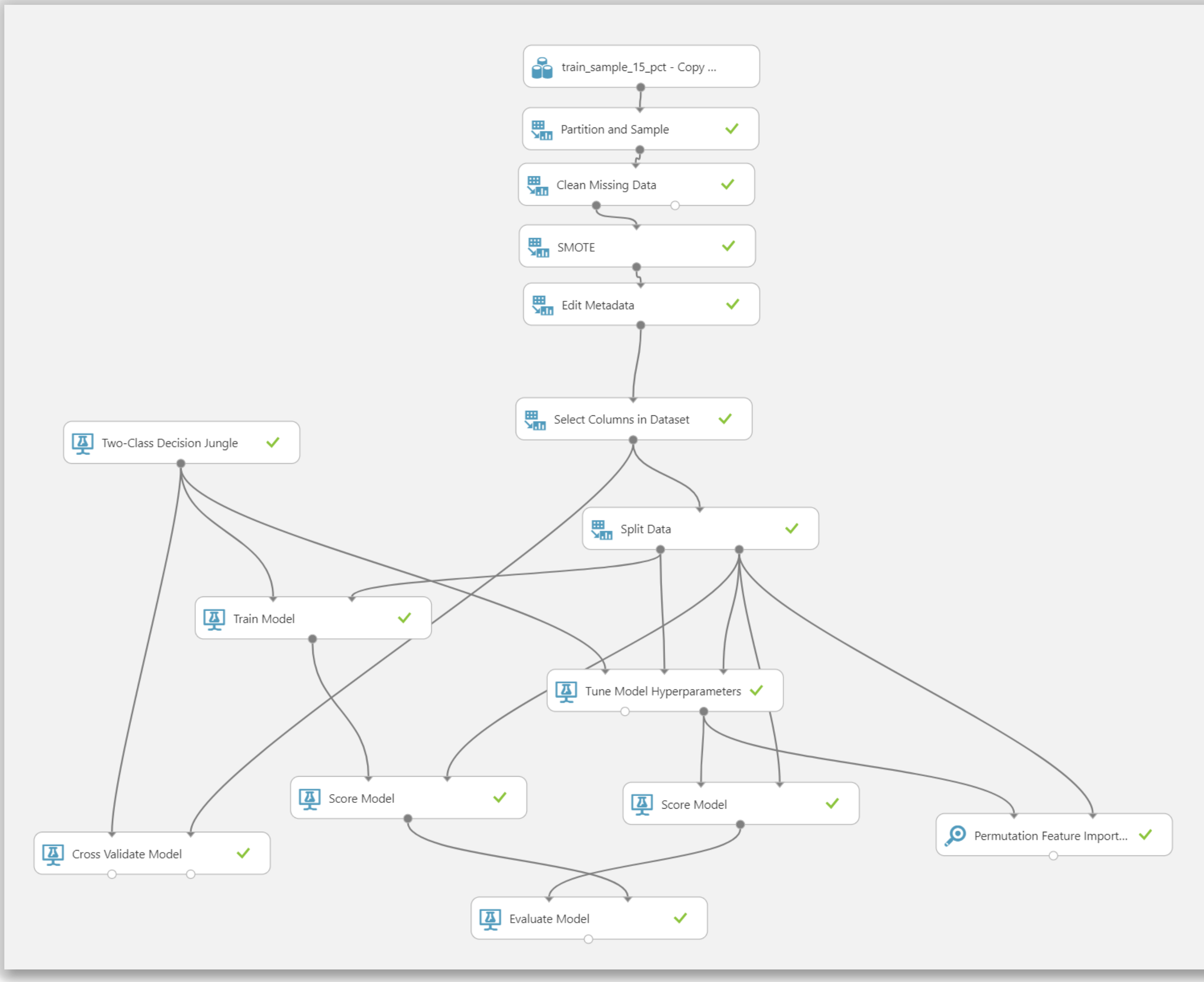
1. Search for the **Two Class Decision Jungle** module. Drag this module onto the canvas. Set the Properties if this module as follows:



2. Search for the **Train Model** module. Drag this module onto the canvas. On the Properties pane, launch the column selector and select the is_attributed column (this is the label column we are trying to predict):
   - **Column Selector**: is_attributed

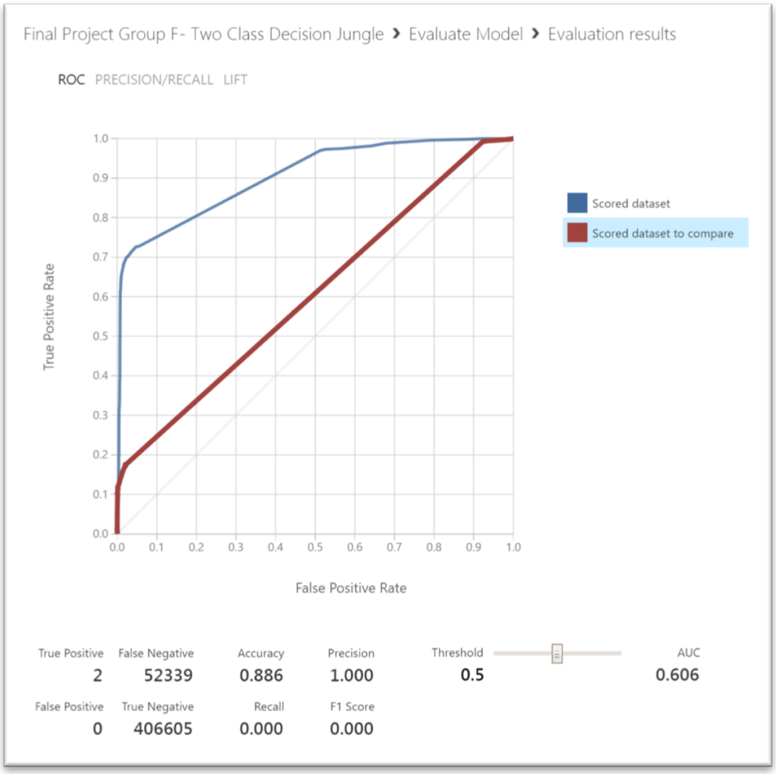
3. Connect the **Untrained Model** output port of the **Two Class Decision Jungle** module to the **Untrained Model** input port of the **Train Model** module. Connect the **Results dataset1** output port of the **Split Data** module to the **Dataset input** port of the **Train model** module.
4. Search for the **Score Model** module and drag it onto the canvas.
5. Connect the **Trained Model** output port of the of the **Train Model** module to the **Trained Model** input port of the **Score Model** module. Connect the **Results dataset2** output port of the **Split Data** module to the **Dataset** port of the **Score Model** module.
6. You will now improve the machine learning model by sweeping the parameter space.
   Search for the **Tune Model Hyperparameters** module. Drag this module onto the canvas and connect the experiment modules as follows:
   a. Connect the **Untrained model** output port of the **Two-Class Decision Jungle** module to the **Untrained model** input port of the **Tune Model Hyperparameters** module.
   b. Connect the **Results dataset1** output port of the **Split Data** module to the **Training dataset** input port of the **Tune Model Hyperparameters** module.
   c. Connect the **Results dataset2** output port of the **Split Data** module to the **Optional test dataset** input port of the **Tune Model Hyperparameters** module.
7. Search for the **Score Model** module again and drag another one onto the canvas and place it under the Tune Model Hyperparameters module.
8. Connect the **Trained best model** (right-hand) output of the **Tune Model Hyperparameters** module to the **Trained Model** input of the **2nd Score Model** module. Connect the **Results dataset2** output port of the **Split Data** module to the **Dataset** port of the **2nd Score Model** module.
9. Click the **Tune Model Hyperparameters** module to expose the **Properties** pane. Set the properties as follows so that 10 combinations of parameters are randomly tested to predict the **is_attributed** variable:
   - **Specify parameter sweeping mode:** Random sweep
   - **Maximum number of runs on random sweep**: 10
   - **Random seed:** 12345
   - **Column Selector:** is_attributed
   - **Metric for measuring performance for classification:** AUC
   - **Metric for measuring performance for regression:** Root of mean squared error

10. Search for the **Permutation Feature Importance** module and drag it onto the canvas. Connect the **Trained best model** (right-hand) output of the **Tune Model Hyperparameters** module to the **Trained model** input port of the **Permutation Feature Importance** module. Connect the **Results dataset2** output port of the **Split Data** module to the **Dataset** port of the **Test data** input port of the **Permutation Feature Importance** module. Set the properties as follows:
    - **Classification:** Accuracy
    - **Random seed**: 12345
11. Search for the **Cross Validate Model** module. Drag this module onto the canvas. Connect the **Untrained model** output from the **Two-Class Decision Jungle** module to the **Untrained model** input port of the **Cross Validate Model** module. Connect the **Results dataset** output port of the **Select Columns in Dataset** module to the **Dataset input** port of the **Cross Validate Model** module.
12. Click the **Cross Validate Model** module to expose the Properties pane. Set the properties as follows:
    - **Column Selector**: is_attributed
    - **Random seed**: 3467
13. Search for the **Evaluate Model** module and drag it onto the canvas. Connect the Scored Dataset output port of the **1st Score Model** module to the right hand **Scored dataset** input port (scored dataset to compare) of the **Evaluate Model** module. Then Connect the **Scored Dataset** output port of the **2nd Score Model** module to the left hand **Scored dataset** input port of the **Evaluate Model** module.
14. Your experiment should now resemble the following:

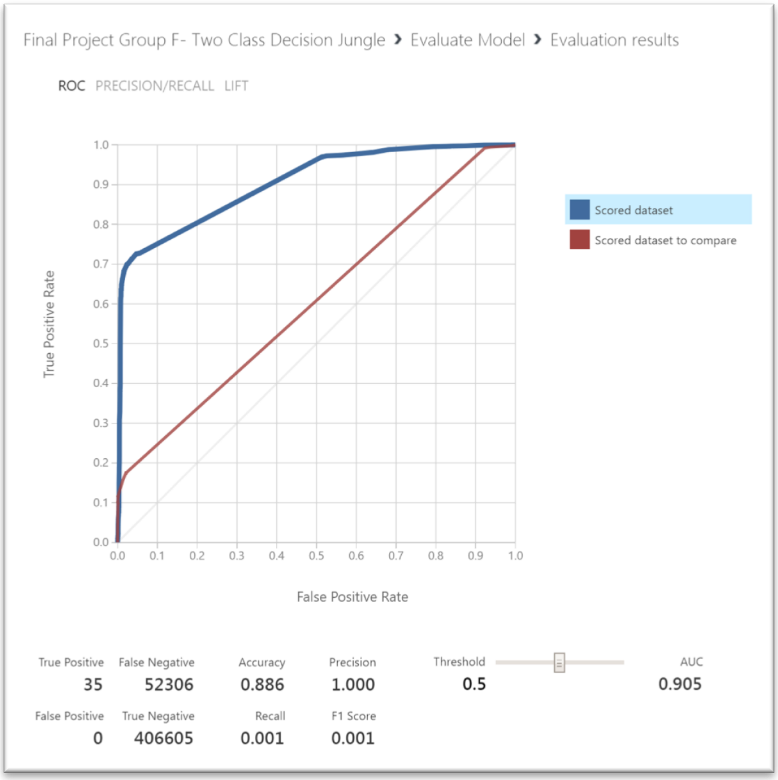


15. Save and run the experiment. When the experiment is finished, visualize the **Evaluation Result** port of the **Evaluate Model** module and review the ROC curve and performance statistics for the model as shown below.

    First let's examine the results without Tune Model Hyperparameters. Click on **Scored dataset to compare** to display the results **without Tune Model Hyperparameters (red line).** The result should look like the image below:



Next click on **Scored dataset** to display the results **with Tune Model Hyperparameters (blue line):**



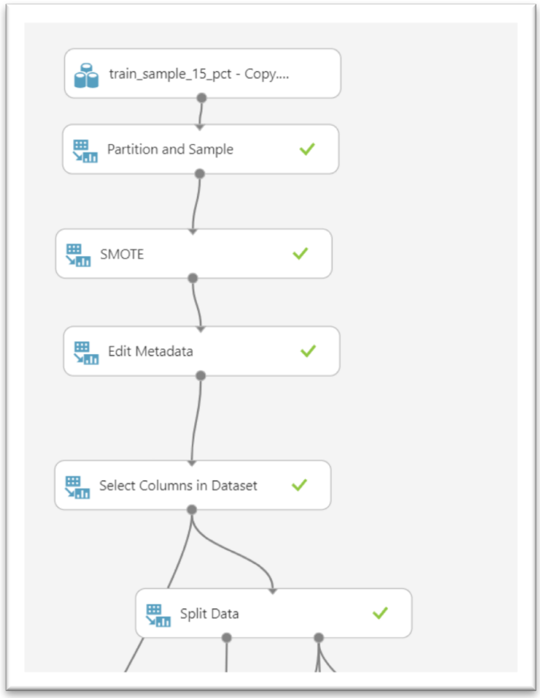
Note that the **AUC without Tune Model Hyperparameter** (red line) is **0.606** and **with Tune Model Hyperparameters** (blue line) it has improved significantly to **0.905**. Also, notice that **Precision** is at 1.0 and the number of **FP** is zero.

Next, we will build a different classification model to see if we can improve these results

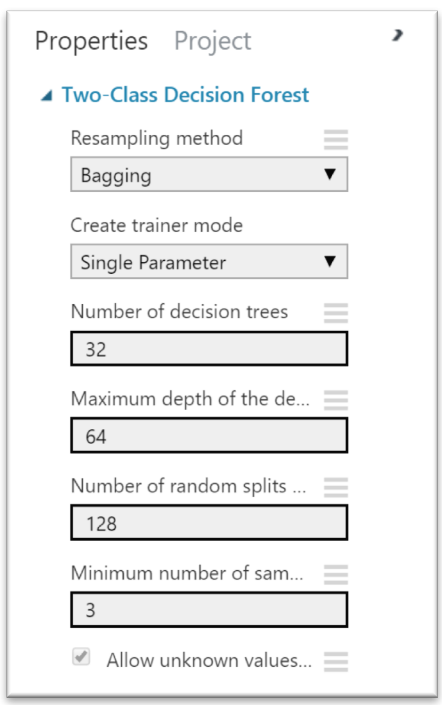
Build Model #2: Two-class Decision Forest

1. Create a new EXPERIMENT and name it Two-class Decision Forest.
2. Repeat STEP 1 and STEP 2 from above but omit the clean data module this time.

This portion of your experiment should look like this:



3. Search for the **Two Class Decision Forest** module. Drag this module onto the canvas. Set the Properties of this module as follows:

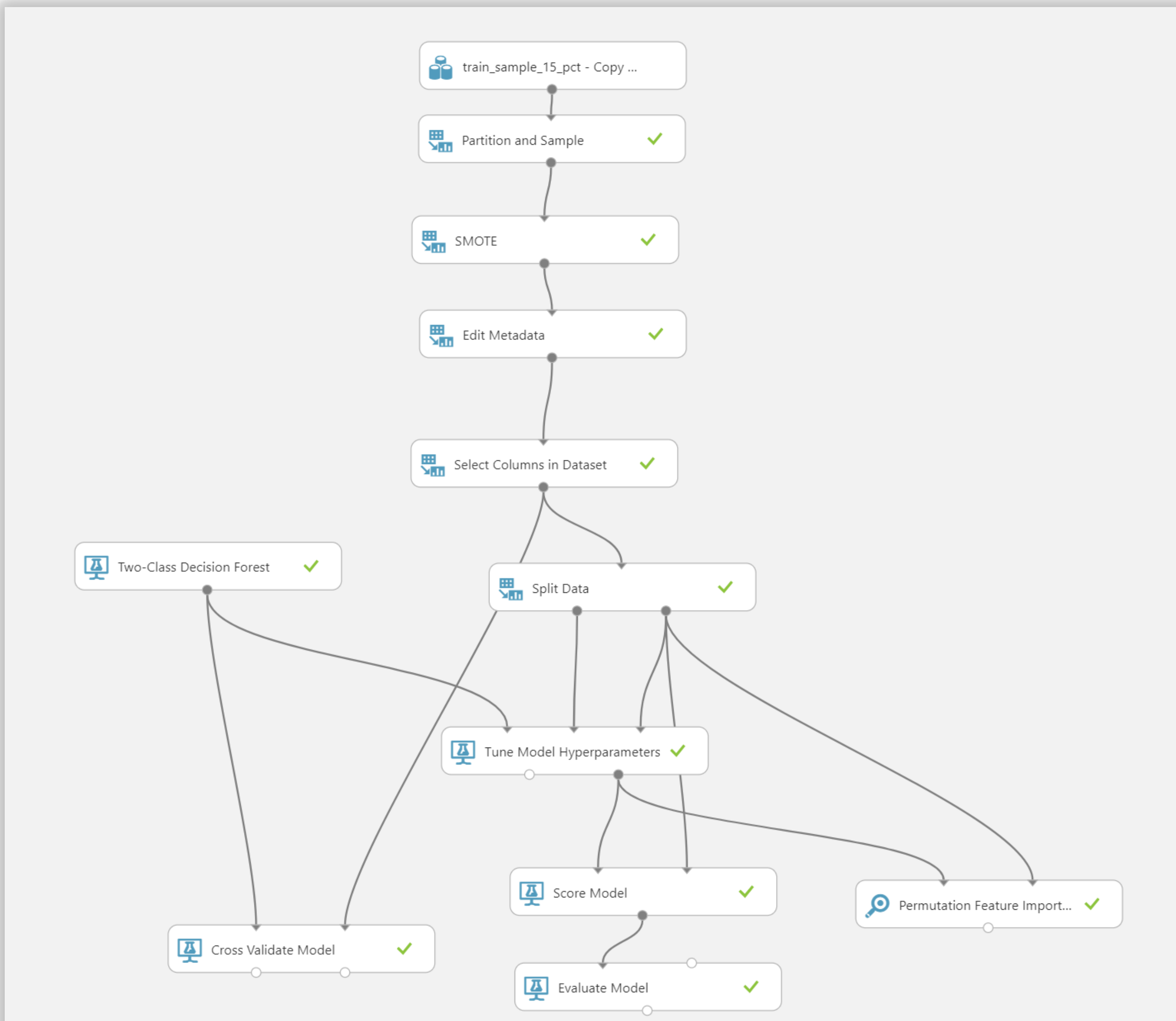


4. Search for the **Tune Model Hyperparameters** module. Drag this module onto the canvas and connect the experiment modules as follows:
    a. Connect the **Untrained model** output port of the **Two-Class Decision Forest** module to the **Untrained model** input port of the **Tune Model Hyperparameters** module.
    b. Connect the **Results dataset1** output port of the **Split Data** module to the **Training dataset** input port of the **Tune Model Hyperparameters** module.
    c. Connect the **Results dataset2** output port of the **Split Data** module to the **Optional test dataset** input port of the **Tune Model Hyperparameters** module.
5. Click the **Tune Model Hyperparameters** module to expose the **Properties** pane. Set the properties as follows:
    - **Specify parameter sweeping mode:** Random grid
    - **Maximum number of runs on random sweep**: 30
    - **Random seed:** 12345
    - **Column Selector:** is_attributed
    - **Metric for measuring performance for classification:** Precision
    - **Metric for measuring performance for regression:** Root of mean squared error
6. Search for the **Score Model** module, drag it onto the canvas and place it under the Tune Model Hyperparameters module. Connect the **Trained best model** (right-hand) output of the **Tune Model Hyperparameters** module to the **Trained Model** input of the **Score Model** module. Connect the **Results dataset2** output port of the **Split Data** module to the **Dataset** port of the **Score Model** module.
7. Search for the **Permutation Feature Importance** module and drag it onto the canvas. Connect the **Trained best model** (right-hand) output of the **Tune Model Hyperparameters** module to the **Trained model** input port of the **Permutation Feature Importance** module. Connect the **Results dataset2** output port of the **Split Data** module to the **Dataset** port of the **Test data** input port of the **Permutation Feature Importance** module. Set the properties as follows:
    - **Classification:** Precision
    - **Random seed**: 12345
8. Search for the **Cross Validate Model** module. Drag this module onto the canvas. Connect the **Untrained model** output from the **Two-Class Decision Jungle** module to the **Untrained model** input port of the **Cross Validate Model** module. Connect the **Results dataset** output port of the

Select Columns in Dataset module to the **Dataset input** port of the **Cross Validate Model** module.

9. Click the **Cross Validate Model** module to expose the Properties pane. Set the properties as follows:
    - **Column Selector**: is_attributed
    - **Random seed**: 3467
10. Search for the **Evaluate Model** module and drag it onto the canvas. Connect the Scored Dataset output port of the **Score Model** module to the left hand Scored dataset input port of the Evaluate Model module.

    Your experiment should now resemble the following:



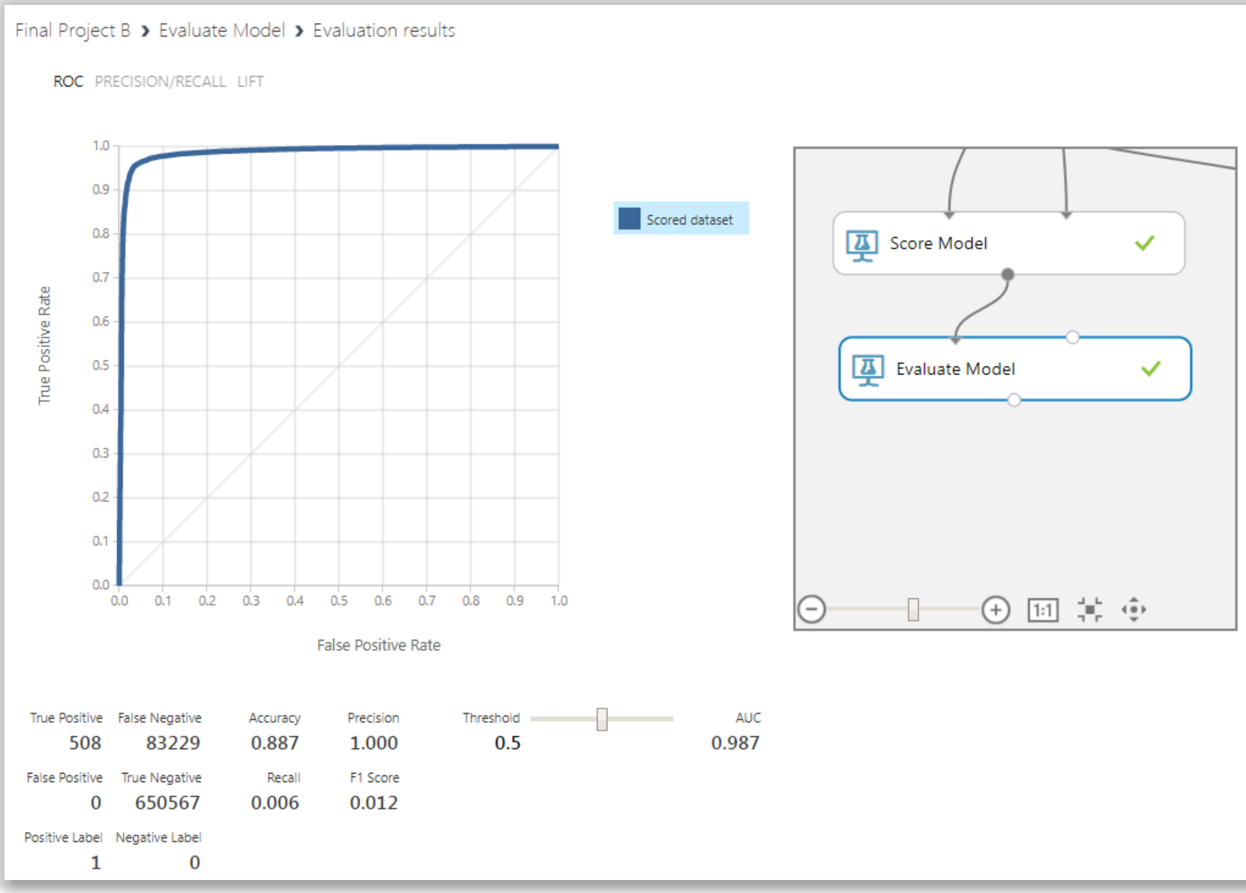
11. Save and run the experiment. When the experiment is finished, click on the **Evaluation Results by Fold** output port of the **Cross Validate Model** and select **Visualize.** These results look like the following:

Final Project - Two Class Decision Forest › Cross Validate Model › Evaluation results by fold

rows 12 columns 10

Fold Number	Number of examples in fold	Model	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
0	244768	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.885496	0	0	0	0.952675	0.324843	8.708544
1	244768	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.886578	0	0	0	0.950305	0.320513	9.359797
2	244768	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.886738	0	0	0	0.960451	0.316589	10.386319
3	244768	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.885467	0	0	0	0.951249	0.325006	8.677798
4	244768	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.884981	0	0	0	0.941921	0.32598	8.658941
5	244768	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.886141	0	0	0	0.957163	0.320884	9.484623
6	244768	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.886619	0	0	0	0.951296	0.319663	9.578603
7	244768	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.886064	0	0	0	0.950825	0.323487	8.79126
8	244768	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.884969	0	0	0	0.948392	0.327692	8.185711
9	244769	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.886591	0	0	0	0.94141	0.327767	7.301329
Mean	2447681	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.885964	0	0	0	0.950569	0.323242	8.913293
Standard Deviation	2447681	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.000688	0	0	0	0.00587	0.003706	0.846102

12. Scroll to the bottom of the page, notice that **the Accuracy, Recall and AUC** values in the folds are not that different from each other. The values in the folds are close to the Mean and the **Standard Deviation** is much smaller than the **Mean.** These consistent results across the folds indicate that the model is insensitive to the training and test data chosen and should generalize well.
13. Next visualize the **Evaluation Result** port of the **Evaluate Model** module and review the ROC curve and performance statistics for the model as shown below:

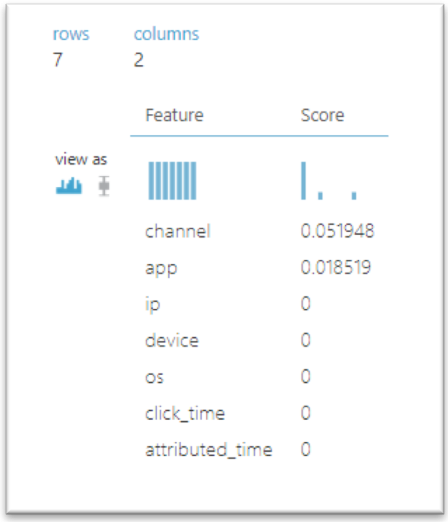


Note that the **AUC** for this model is **0.987** which is better than the Decision Tree Jungle Model's AUC of 0.905. **Precision** is at 1.0 and the number of **FP** is at zero.

Step 4: Improve the model

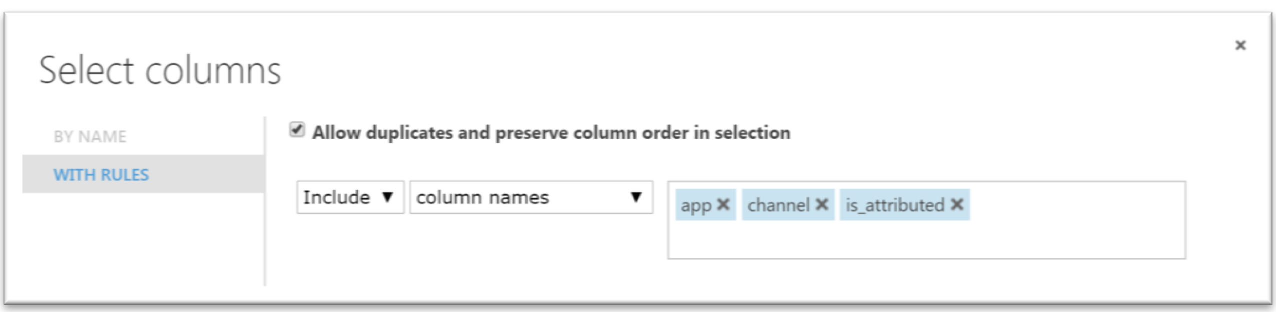
You will now improve model performance by pruning less important features by following these steps:

1. Visualize the output of the **Permutation Feature Importance** module. The upper portion of the list produced should resemble the following:



Notice that all but top 2 features have zero importance. This means that the channel and app features are most important for building our model. Therefore, we will create a new experiment with those two features only in order to improve our model's performance.

2. With your Decision Tree Forest experiment open, click on **SAVE AS** at the bottom of the screen to save a copy of this experiment under the name Running with Permutation Decision
3. Click on **Select Columns in Dataset (Project Columns) module**, and in the properties pane, launch the column selector. Modify the module to include only the following columns as shown below:



4. Save and run the experiment. When the experiment is finished, visualize the output of the **Score Model** it should resemble the following:

Running with Permutation decision › Score Model › Scored dataset

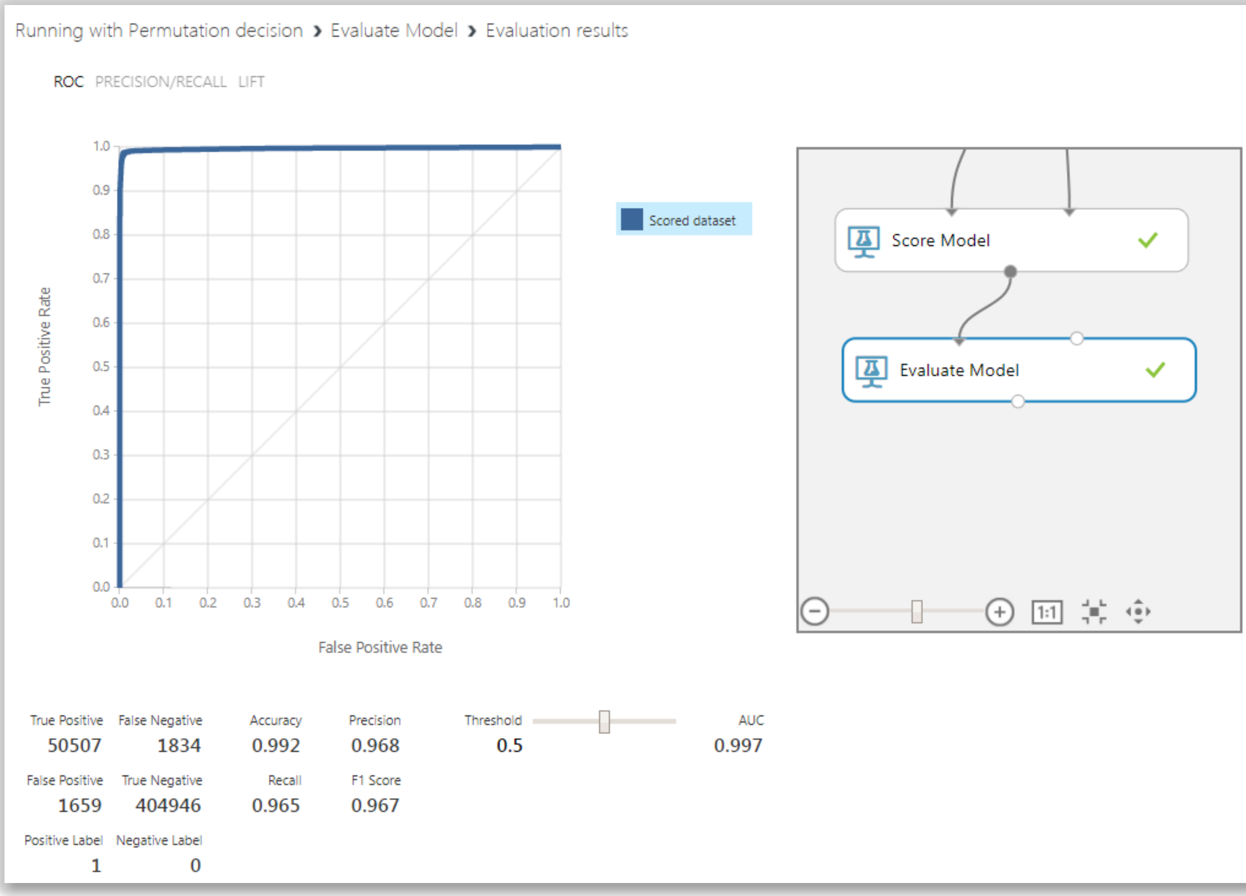
rows	columns
458946	5

view as

app	channel	is_attributed	Scored Labels	Scored Probabilities
12	245	0	0	0.000187
13	210	1	1	0.881142
2	205	0	0	0.000737
35	21	1	1	0.928787
12	178	0	0	0.001331
15	111	0	0	0.002389
8	145	0	0	0.002046
93	371	0	0	0.008463
8	145	0	0	0.002046
18	107	0	0	0.000869
2	469	0	0	0.000867
11	347	1	1	1
12	219	0	0	0.010909
8	145	0	0	0.002046
12	409	0	0	0.00035
24	105	0	0	0.004478
15	315	0	0	0.000652
33	230	1	1	0.988169
12	265	0	0	0.000146
12	140	0	0	0.001015
2	469	0	0	0.000867
32	21	0	0	0.486852

Notice how the **Scored Labels** are predicted correctly against the **is_attributed** column

5. Next visualize the output of the **Evaluate Model** module and examine the ROC curve and summary statistics as shown below:



Notice that by keeping only the 2 features (channel and app) we were able to improve **AUC** from 0.987 to 0.997. However, **Precision** has decreased slightly from 1.0 to 0.968 and the number of **False Positives** has increased from 0 to 1659.

Selecting the appropriate performance metrics

In the classification model, there are four main measures: Accuracy, Precision, Recall and F1 score which are part of what is called the Confusion Matrix. It is very important to correctly choose the appropriate metric. As indicated in the table below, both Precision and Recall work well when there is an uneven class distribution, as is our case.

		CLASS DISTRIBUTION	
		EVEN	UNEVEN
COST	FN Cost More	Recall	Recall
	Same Cost	Accuracy or F1 Score	F1 Score
	FP Cost More	Precision	Precision

For our project, **False Positives (FP),** which indicate the model predicted an app was downloaded when in fact it wasn't, are more important than False Negatives (FN) because focusing on minimizing False Positives will help us save money and better target advertisements. Thus, given that we have an uneven dataset and False Positives cost more, for our project, **Precision** is the key metric.

Therefore, next, we will try to **improve Precision**.

6. Move the **Threshold** slider to right to increase it from 0.5 to 0.8. Notice that **Precision** increased to 0.992, **FP** decreased from 1,659 to 377 and **FN** increased from 1,834 to 5,142. However, as mentioned above, for the purposes of this experiment, we care most about maximizing Precision and minimizing False Positives. Therefore, at this point we are quite satisfied with the results our model produced and can stop here.

References

1. Dataset (1GB) https://drive.google.com/open?id=1UHEOMbgsIjl-c2LOUghI9g4g3wMC2IhU
2. Original (7GB) dataset link https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data
3. Github Link https://github.com/ngupta8/CIS-5560
4. https://blogs.msdn.microsoft.com/andreasderuiter/2015/02/09/performance-measures-in-azure-ml-accuracy-precision-recall-and-f1-score/
5. https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/machine-learning-initialize-model-classification
6. https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice
7. https://docs.databricks.com/spark/latest/mllib/binary-classification-mllib-pipelines.html