Fraud Transaction Detection Using Azure Machine Learning

By Bhagyasree Bhagwat, Niklas Melcher, Priyanka Purushu

Instructor: Jongwook Woo, 05/19/2017

Overview

In this tutorial, you will train and evaluate a classification model. Classification is one of the fundamental machine learning methods used in data science. Classification models enable you to predict classes or categories of a label value. Classification algorithms can be two-class methods, where there are two possible categories or multi-class methods. In our project since we just have 2 category that is fraud or genuine transaction, we will use two-class methods such as

- Two-class Logistic Regression
- Two-class Decision Forest
- Two-class Decision Jungle
- Two-class support Vector Machine

Like regression, classification is a supervised machine learning technique, wherein models are trained from labeled cases. As discussed earlier, in this tutorial you will use the data set provided to determine if the mobile money transaction made was a fraud or genuine transaction. The steps in this process include:

- Investigate relationships in the data set with visualization using Python code.
- Retaining attributes that are important for the prediction and Normalize the data.
- Create a machine learning classification model.
- Improve the model by Tune model hyperparameters.
- Cross validate the model.
- Evaluate the model.

What You'll Need

To complete this lab, you will need the following:

- An Azure ML account
- A web browser and Internet connection
- Python
- The dataset on Fraud Transaction to be downloaded from Kaggle – https://www.kaggle.com/ntnu-testimon/paysim1

Preparing and Exploring the Data

This project aims at analyzing data and providing insights on Financial Fraud Detection using Azure ML. The dataset contains a sample of real transactions extracted from one month of financial logs from a mobile money service. The original logs were provided by the multinational company, who is the provider of the mobile financial service which is currently running in more than 14 countries all around the world. This dataset is scaled down to 1/4 of the original dataset and it is created just for Kaggle.

Upload the Data Set

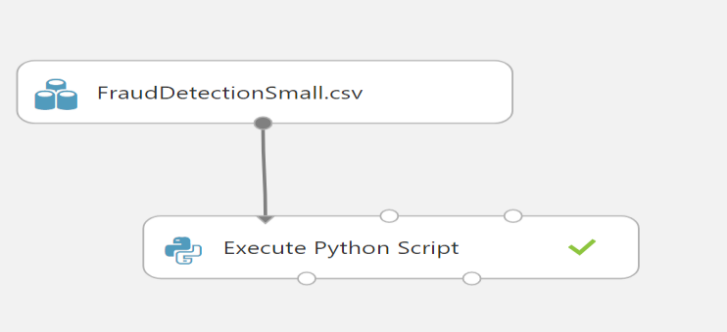
The full financial fraud detection data set requires a lengthy model training time because the size of the file is approximately 470 MB. To save this effort, we have uploaded only a subset containing of just 5171 rows.

1. If you have not already done so, open a browser and browse to https://studio.azureml.net. Then sign in using the Microsoft account associated with your Azure ML account.
2. Create a new blank experiment, and give it the title **Fraud Detection Small** .
3. With the **Fraud Detection Small** experiment open, at the bottom left, click **NEW**. Then in the **NEW** dialog box, click the **DATASET** tab as shown in the following image.
4. Click **FROM LOCAL FILE**. Then in the **Upload a new dataset** dialog box, browse to select the **frauddetectionsmall.csv** file.
5. Enter the following details as shown in the image below, and then click the **OK** icon.
    - **This is a new version of an existing dataset**: Unselected
    - **Enter a name for the new dataset**: frauddetectionsmall (Clean)
    - **Select a type for the new dataset**: Generic CSV file with a header (.csv)
    - **Provide an optional description**: Fraud Detection Transaction.
6. Wait for the upload of the dataset to be completed, and then on the experiment items pane, expand **Saved Datasets** and **My Datasets** to verify that the **frauddetectionsmall (Clean)** dataset is listed.

Visualize the Data with Python

In this exercise, you will create custom Python code to visualize the data set and examine the relationships. This data set has 11 feature columns, with both numeric and categorical (string) features. The label column is titled **isFraud**. The label column can have two values “0” and “1”. Having two values in the label makes this a two-class or binary classification problem.

1. Drag the **frauddetectionsmall (Clean)** and onto the canvas.
2. Search for the **Execute Python Script** module and drag it onto the canvas. Connect the output of the data set to the left input (**Dataset1**) port of the **Execute Python Script** module. At this point your experiment should resemble the following:



3. Click the **Execute Python Script** module, and in the Properties pane, replace the existing code with the following code, which you can copy from the **Fraudtrasactionvis.py** file in the github:

```
import matplotlib
matplotlib.use('agg')


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.graphics.boxplots as sm
from pandas.tools.plotting import scatter_matrix
import pandas.tools.rplot as rplot
def azureml_main(frame1):
    Azure = True


## Create a series of bar plots for the various levels of the
## string columns in the data frame by readmi_class.
    result = frame1.info()
    result2 = frame1.isnull().values.any()
    print("IS NULL", result2)


   #for i, group in frame1.groupby('type'):
        #fig = plt.figure()
        #fr = group['isFraud']
        #group.plot(x=i, y=fr, title=str(i))
        #fig.savefig('bar_' + i +'.png')
    del frame1['nameOrig'], frame1['nameDest']
    #gr = frame1.groupby(['type'])
```

```
fig1 = plt.figure(0)
f, ax = plt.subplots(1, 1, figsize=(8, 8))
frame1.type.value_counts().plot(kind='bar', title="Transaction type", ax=ax, figsize=(8,8))
plt.show(fig1)
fig2 = plt.figure(1)
cond = (frame1['isFraud'] >= 1)
taf = frame1[cond].type.value_counts().plot(kind='bar',  title="Fraud transactions grouped by type")
fig2.savefig('fig2.png')
plt.show(taf)
fig3 = plt.figure(2)
cond2 = (frame1['isFraud'] < 1)
taf2 = frame1[cond2].type.value_counts().plot(kind='bar',  title="No fraud transactions grouped by type")
fig3.savefig('fig3.png')
plt.show(taf2)
#fraud['type'] = fraud['type'].apply(convert)
#fraud1 = normalize(fraud)
fig4 = plt.figure(3)
medianprops = dict(linestyle='-', linewidth=2, color='blue')
bx1 = frame1[cond2].boxplot(column=['oldbalanceDest', 'newbalanceDest'], by='isFraud')
bx2 = frame1[cond2].boxplot(column=['oldbalanceOrg', 'newbalanceOrig'], by='isFraud')
bx3 = frame1[cond2].boxplot(column=['amount'], by='isFraud')
#bx4 = fraud[cond].boxplot(column=['type'], by='isFraud', medianprops=medianprops)
fig4.savefig('fig4.png')
plt.show(bx3)
#hist1 = frame1[cond2].plot.hist(by='isFraud', stacked=True, bins=20)
#fig5 = plt.figure(3)
fig5 = plt.figure(4)
bx4 = frame1[cond].boxplot(column=['oldbalanceDest', 'newbalanceDest'], by='isFraud')
bx5 = frame1[cond].boxplot(column=['oldbalanceOrg', 'newbalanceOrig'], by='isFraud')
bx6 = frame1[cond].boxplot(column=['amount'], by='isFraud')
plt.show(bx4)
fig5.savefig('fig5.png')
plt.show(bx5)
return result
```

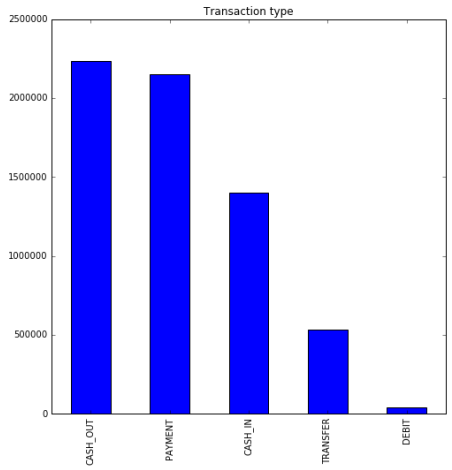
Tip: To copy code in a local code file to the clipboard, press **CTRL+A** to select all of the code, and then press **CTRL+C** to copy it. To paste copied code into the code editor in the Azure ML **Properties** pane, press **CTRL+A** to select the existing code, and then press **CTRL+V** to paste the code from the clipboard, replacing the existing code.

WARNING!: Ensure you have a Python return statement at the end of your azureml_main function; for example, return frame1. Failure to include a return statement will prevent your code from running and may produce an inconsistent error message.

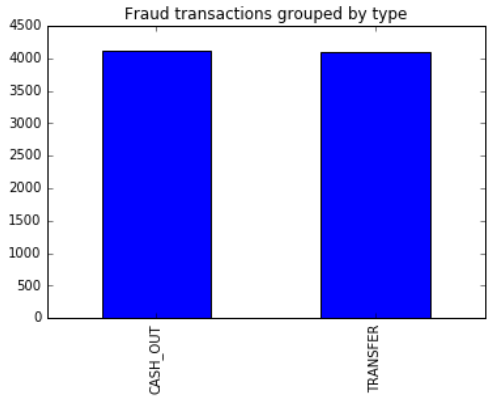
This code creates bar plots for the categorical variables and box plots for numeric variables in the data set with the following steps:

- Create bar plots for *type* categorical column in the data frame.
- Create box plots for *oldbalanceDest* and *newbalanceDest* in the data frame. The box plots are grouped by **isFraud**

4. Save and run the experiment. When the experiment has finished running, visualize the output from the **Python Device** port.
5. You will see a conditioned bar plot for each of the categorical features. Examine these bar plots to develop and understanding of the relationship between these features and the label value.
6. The below graph shows the amount of transactions grouped by the type. We see that most of the transactions are made with CASH_OUT and PAYMENT.

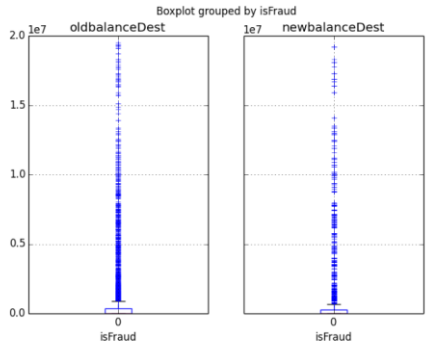


7. Next, we want explore which transaction types are vulnerable to fraud. The below bar plot shows clearly that fraud transactions are only made with the type CASH_OUT and TRANSFER.



8. Locate the box plots of each numeric variable conditioned on the levels of the label. Examine these box plots to develop and understanding of the relationship between these features and the label values.

9. Locate the box plot of **oldbalanceDest** and **newbalanceDest** grouped by label **isFraud** which should resemble the figure below.

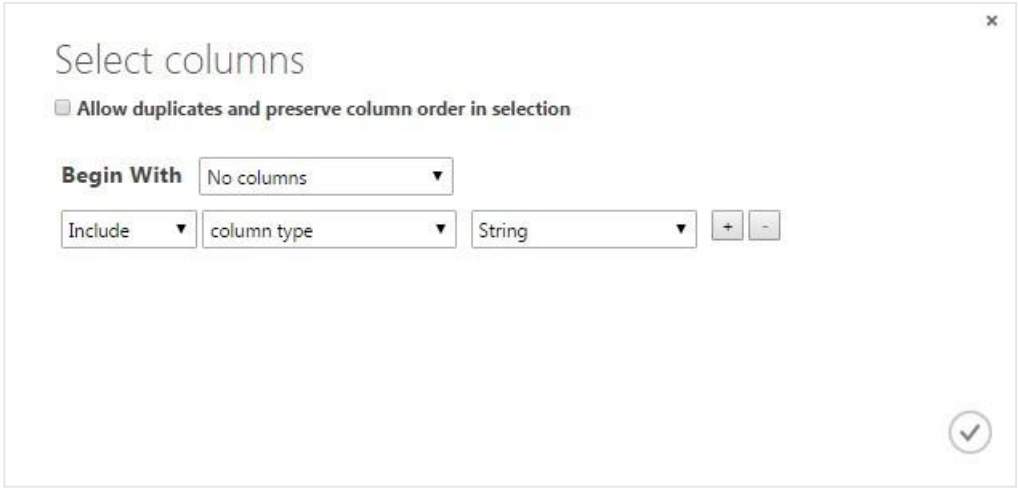


Building a Classification Model

Now that you have investigated the relationships in the data you will build, improve and validate a machine learning model.

Create a New Model

1. If you are working with Python, you need to add a **Edit Metadata** module to your experiment by following steps a, b and c below.
    a. Search for the **Edit Metadata** and drag it onto the canvas. Connect the output of the **frauddetectionsmall(Clean)** data set to the input of the **Edit Metadata**.
    b. Click the **Edit Metadata** and in the properties pane, launch the **Column Selector**. Select all string columns as shown:



    c. In the **Categorical** drop down list, select **Make Categorical**.

2. Search for the **Select Columns in Dataset** module and drag it onto your canvas. Connect the **Results Dataset** output of the **Edit Metadata** module to the input port of the **Select Columns in Dataset module**.

Note: With the **Select Columns in Dataset** module, you will remove features from the data set found to be poor separators of the label cases during visualization of the dataset.

3. With the **Select Columns in Dataset** module selected, in the properties pane, launch the column selector, and exclude the following columns:
    - step
    - nameDest
    - nameOrig
    - isFlaggedFraud

4. Search for the Normalize Data module. Drag this module onto your experiment canvas. Connect the Results dataset output of the Select Columns in Dataset module to the **Dataset** input port of the **Normalize Data** module. Set the **Properties** of the **Split Data** module as follows:
    - Transformation method: ZScore
    - Columns to transform: amount, oldbalanceOrg, newbalanceOrig, newbalanceDest, oldbalanceDest

5. Search for the **Split Data** module. Drag this module onto your experiment canvas. Connect the **Results dataset** output port of the **Normalize Data** module to the **Dataset** input port of the **Split Data** module. Set the **Properties** of the **Split Data** module as follows:
    - **Splitting mode:** Split Rows
    - **Fraction of rows in the first output:** 0.8
    - **Randomized Split:** Checked
    - **Random seed:** 1256
    - **Stratified Split:** False

6. Search for the **Two Class Decision Forest** module. Make sure you have selected the regression model version of this algorithm. Drag this module onto the canvas. Set the Properties if this module as follows:
    - **Resampling method:** Bagging
    - **Create trainer mode:** Parameter Range
    - **Number of decision trees:** 40
    - **Maximum depth of the decision trees:** 32
    - **Number of random Splits per node:** 128
    - **Minimum number of samples per leaf node:** 4

7. Search for the **Tune Model Hyperparameters** module. Drag this module onto the canvas in place of the **Train Model** module you removed, and reconnect the experiment modules as follows:
    - 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.
    - Connect the **Results dataset1** output port of the **Split Data** module to the **Training dataset** input port of the **Tune Model Hyperparameters** module.
    - Connect the **Results dataset2** output port of the **Split Data** module to the **Optional test dataset** input port of 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.

8. Click the **Tune Model Hyperparameters (Sweep Parameters)** module to expose the **Properties** pane. Set the properties as follows so that 40 combinations of parameters are randomly tested to predict the **isFraud** variable:
    - **Specify parameter sweeping mode:** Random grid
    - **Maximum number of runs on random sweep**: 40
    - **Random seed:** 4567
    - **Column Selector:** isFraud
    - **Metric for measuring performance for classification:** Recall
    - **Metric for measuring performance for regression:** Mean absolute error (this doesn't matter for a classification model)

9. Search for the **Cross Validate Model** module. Drag this module onto the canvas. Connect the **Untrained model** output from the **Two-Class Decision Forest** 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
10. Click the **Cross Validate Model** module to expose the Properties pane. Set the properties as follows:
    - **Column Selector**: isFraud
    - **Random seed**: 3467

11. When the experiment has run click on the **Evaluation Results by Fold** output port of the **Cross Validation Model** and select **Visualize**. Scroll to the right and note the **Accuracy, Recall** and **AUC** columns (the first, third and fifth numeric columns from the left). Scroll to the bottom of the page, passed the results of the 10 folds of the cross validation and examine the **Mean** value row. These results look like the following:

6	517	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	1	1	1	1	1	0.004903	86.253301
7	517	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.998066	1	0.8	0.888889	0.999609	0.00587	89.226435
8	517	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	1	1	1	1	1	0.002404	93.258948
9	518	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.996139	1	0.333333	0.5	0.831068	0.062788	-76.325706
Mean	5171	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.996519	0.966667	0.596667	0.700274	0.936186	0.035266	22.66445
Standard Deviation	5171	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.002378	0.105409	0.290678	0.235266	0.069773	0.031654	71.296708

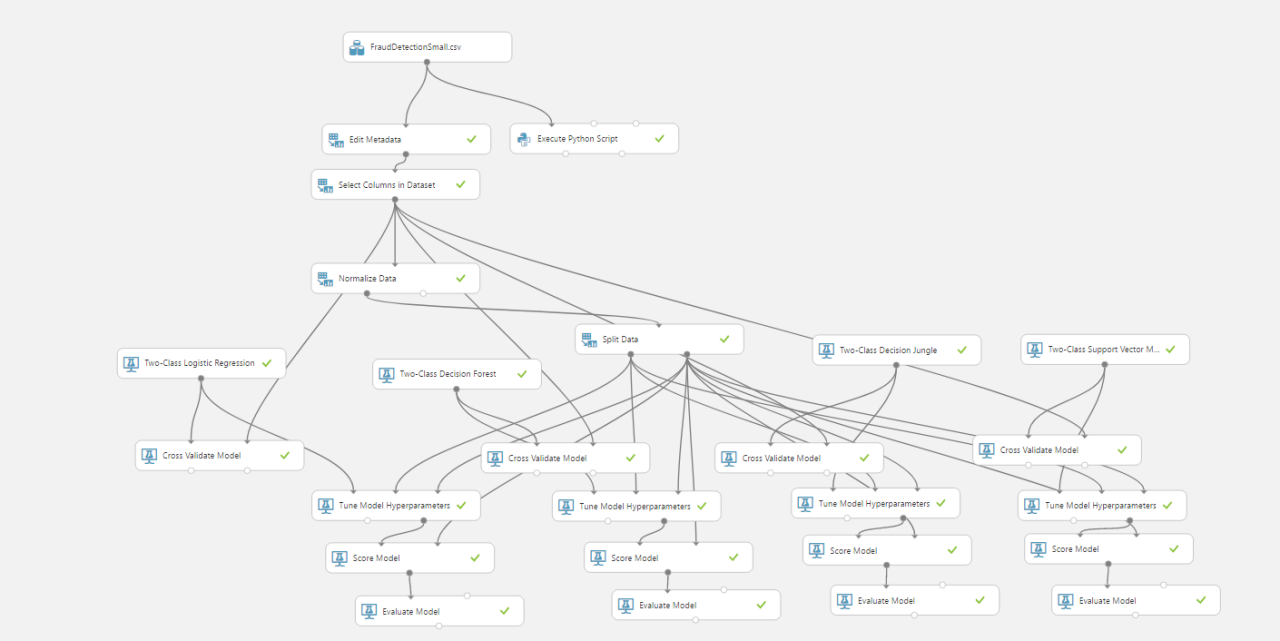
Notice that the **Accuracy, Recall** and **AUC** values in the folds are not that different from each other. The values in the folds are close to the **Mean**. Further, the **Standard Deviation** is much smaller than the **Mean**. These consistent results across the folds indicate that the model is insensitive to the training and test data chosen and should generalize well

12. Search for the **Score Model** module and drag it onto the canvas.
13. Connect the **trained best model** output port of the of the **Tune Model Hyperparameter** 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.

14. 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.

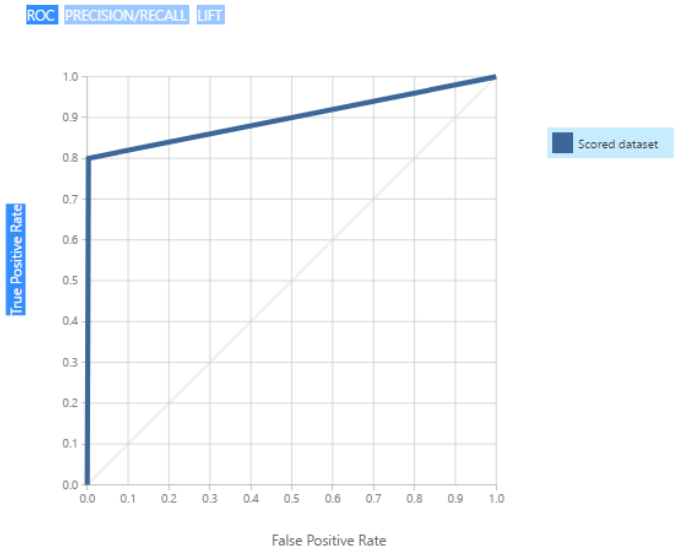
15. In this project, we have used the following algorithm mentioned below. Perform all the steps from 6 to 13 on all the remaining 3 algorithms.
    - Two-class Logistic Regression
    - Two-class decision Forest
    - Two-class decision Jungle
    - Two-class support vector machine

16. Finally the project would look like the following:



17. 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.
18. Examine this ROC curve. Notice that the bold blue line is well above the diagonal grey line, indicating the model is performing significantly better than random guessing. The AUC is 0.897

(see below), which is significantly more than 0.5 obtained by random guessing.



19. Next examine the performance statistics for two-class decision forest is as shown here:

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
8	2	0.995	0.727	0.5	0.897

False Positive	True Negative	Recall	F1 Score
3	1021	0.800	0.762

Positive Label	Negative Label
1	0

20. From the confusion matrix, we can see that value of FN and FP is less therefore the value of recall and precision is high. Recall = TP/(TP+FN) and Precision = TP/(TP+FP). The Recall, Precision and AUC for the remaining 3 algorithms are as follows.

Model	Accuracy	Precision	Recall
Two Class Logistic Regression	0.991	1.000	0.100
Two Class Decision Forest	0.995	0.727	0.800
Two Class Decision Jungle	0.997	1.000	0.700
Two Class Support Vector Machine	0.993	1.000	0.300

Summary

In this tutorial, you have constructed and evaluated 4 two class or binary classification model. Highlight from the results of this lab are:

- Visualization of the data set can help differentiate features which separate the cases from those that are unlikely to do so.
- Cross validation can indicate how well a model will generalize
- Examining the classification behavior of features can highlight potential performance problems or provide guidance on improving a model.
- Based on the recall, precision, AUC and the time taken to train the model we have come to the conclusion that Two-class decision Forest is the best model.