Machine Learning: Using Python to Predict Credit Card Fraud

Amanda Margaret Rice

American University – Department of Computer Science

CSC-148: Introduction to Computer Science

Professor Ahmed

June 15th, 2022

Credit Card Fraud: The Increase of Contactless Payments and Identify Theft

The covid-19 pandemic has altered consumer behavior through a shift from contact payments to contactless payments. In other words, most payments occur through a technological operating system. This raises the concern of credit card fraud for people all over the world. Credit card fraud is a form of identity theft as it occurs when someone accesses another person's identity to complete a purchase. Credit card fraud seems to be an ubiquitous problem as recent data highlights that nearly 50% of identity theft is fraudulent credit card charges (Boitnott). In the year of 2014, the Federal Bureau of Investigation noted that there were over 20,000 victims affected by fraudulent credit card charges. It is estimated that by 2030, credit card companies will lose nearly $500 billion dollars to credit card fraud (Mullen). Moreover, despite the United States making up less of the global card volume, it is responsible for 36% of fraudulent transactions (Mullen).

The Threat of Becoming a Victim to Identity Theft

The combination of the increased use of technology to make purchases and technological advancements increase the likelihood of fraudulent credit card transactions; therefore, it is important to develop machine learning algorithms that can detect fraudulent transactions to protect credit card holders from identity theft. In short, Machine learning can preclude the rise of fraudulent credit card transactions.

Why Machine Learning is Important to Protecting the Identity of Credit Card Holders

Machine learning builds algorithms that have the ability to "learn and improve from data" (Sridhar) Machine learning is the process of using a foundational dataset to determine which algorithm has the highest accuracy score. In the context of credit card fraud, machine learning is important because it can detect fraudulent transactions.

Data: European Credit Cardholders

The dataset in this study was derived from transactions made by European cardholders in September of 2013. The data used in this study was PCA transformed to ensure the privacy of the credit card holders as it incorporated sensitive information. (ULB). The data can be found here.

Importing the Packages

Shown below are the packages necessary for analyzing the data.

Data Processing

```
import pandas as pd
```

Working with arrays	`import numpy as np`
Visualization	`import matplotlib.pyplot as plt`
Text customization	`from termcolor import colored as cl`
Advanced tools	`import itertools`
Data normalization	`from sklearn.preprocessing import StandardScaler`
Data split	`from sklearn.model_selection import train_test_split`
Decision tree algorithm	`from sklearn.tree import DecisionTreeClassifier`
KNN algorithm	`from sklearn.neighbors import KNeighborsClassifier`
Logistic regression	`from sklearn.linear_model import LogisticRegression`
SVM algorithm	`from sklearn.svm import SVC`
Random forest tree	`from sklearn.ensemble import RandomForestClassifier`
XGBoost algorithm	`from xgboost import XGBClassifier`
Evaluation metric: Confusion Matrix	`from sklearn.metrics import confusion_matrix`
Evaluation metric: Accuracy Score	`from sklearn.metrics import accuracy_score`
Evaluation metric: F1-Score	`from sklearn.metrics import f1_score`

Importing the Data

The data used in this project can be found here.

How to import 'creditcard.csv' data

```
df = pd.read_csv('creditcard.csv')
df.drop('Time', axis = 1, inplace = True)
print(df.head())
```

After entering the above code for importing the data, it will produce something that looks like this:

```
       V1        V2        V3        V4        V5        V6        V7  \
-1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
 1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
-1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
-0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
-1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

       V8        V9       V10  ...       V21       V22       V23       V24  \
 0.098698  0.363787  0.090794  ... -0.018307  0.277838 -0.110474  0.066928
 0.085102 -0.255425 -0.166974  ... -0.225775 -0.638672  0.101288 -0.339846
 0.247676 -1.514654  0.207643  ...  0.247998  0.771679  0.909412 -0.689281
 0.377436 -1.387024 -0.054952  ... -0.108300  0.005274 -0.190321 -1.175575
-0.270533  0.817739  0.753074  ... -0.009431  0.798278 -0.137458  0.141267

       V25       V26       V27       V28  Amount  Class
 0.128539 -0.189115  0.133558 -0.021053  149.62      0
 0.167170  0.125895 -0.008983  0.014724    2.69      0
-0.327642 -0.139097 -0.055353 -0.059752  378.66      0
 0.647376 -0.221929  0.062723  0.061458  123.50      0
-0.206010  0.502292  0.219422  0.215153   69.99      0
```

The columns labeled ‘V1’ through ‘V8’ contain values that were transformed by Principal Component Analysis (PCA). PCA lessens feature redundancy yet strives to preserve the original data through orthogonal transformation (Savasta). Orthogonal transformation is a linear transformation that preserves the meaning of the original data (Savasta). Without PCA transformation, the available data would include information private to the credit card holder. The values under the columns ‘V1’ through ‘V8’ include numerical values of the credit card transactions.

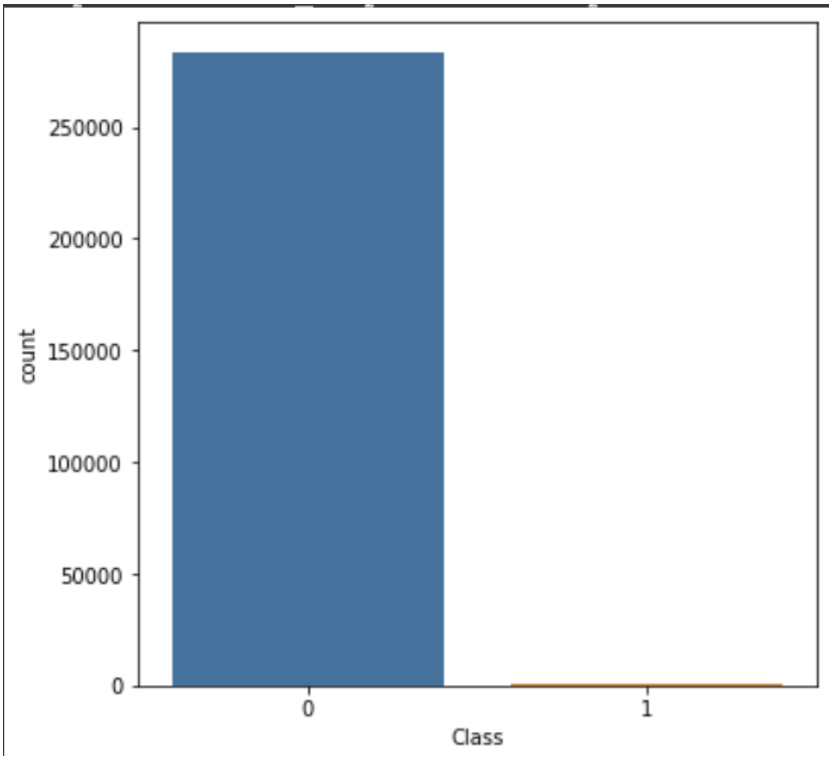
The column labeled ‘Amount’ is the total purchase price of each transaction.

The column labeled ‘Class’ is a binary representation of fraudulent and non-fraudulent cases. The numerical value of ‘0’ in the ‘Class’ column is a representation of non-fraudulent cases whereas the numerical value of ‘1’ in the ‘Class’ column is a representation of the fraudulent transactions.

Histogram

```
df.Class.value_counts()
sns.countplot("Class",data=df)
```

A histogram was computed to illustrate the ‘Class’ column as an effort to better visualize the frequency of the binary representation of fraudulent and non-fraudulent cases in the dataset. As shown in the histogram below using the above code, it is evident that there are lesser fraudulent transactions than non-fraudulent transactions – the great disparity of fraudulent and non-fraudulent cases is because the data used in this machine learning study is from 2013, meaning that the cyber security mechanisms are satisfactory and effective to date.



Exploratory Data Analysis

Exploratory data analysis (EDA) must occur prior to machine learning. Exploratory data analysis is a necessary procedure as it analyzes and interprets a given data set as an effort to check the particular bounds of the data. In other words, it allows the researcher to investigate the data and explore its strengths and weaknesses.

Shape

```
df.shape
(284807, 31)
```

Exploratory data analysis

```
cases = len(df)
nonfraud_count = len(df[df.Class == 0])
fraud_count = len(df[df.Class == 1])
fraud_percentage = round(fraud_count/nonfraud_count*100, 2)
```

Print the EDA code

```
print(cl('CASE COUNT', attrs = ['bold']))
print(cl('----------------------------------------',
attrs = ['bold']))
print(cl('Total number of cases are {}'.format(cases), attrs
```

```
= ['bold']))
print(cl('Number of Non-fraud cases are
{}'.format(nonfraud_count), attrs = ['bold']))
print(cl('Number of Fraud cases are {}'.format(fraud_count),
attrs = ['bold']))
print(cl('Percentage of fraud cases is
{}'.format(fraud_percentage), attrs = ['bold']))
print(cl('-----------------------------------------',
attrs = ['bold']))
```

```
CASE COUNT
-----------------------------------------
Total number of cases are 284807
Number of Non-fraud cases are 284315
Number of Non-fraud cases are 492
Percentage of fraud cases is 0.17
-----------------------------------------
```

As noted before, there is a great disparity between fraudulent and non-fraudulent charges. Out of 284,807 cases, only 492 cases are fraudulent credit card transactions. In other words, the data is not entirely balanced as there is a disparity between fraudulent and non-fraudulent cases. Moreover, the percentage of fraudulent cases is 17%.

Exploratory Data Analysis: Describing the data

```
nonfraud_cases = df[df.Class == 0]
fraud_cases = df[df.Class == 1]
```

Print the code

```
print(cl('CASE AMOUNT STATISTICS', attrs = ['bold']))
print(cl('-----------------------------------------',
attrs = ['bold']))
print(cl('NON-FRAUD CASE AMOUNT STATS', attrs = ['bold']))
print(nonfraud_cases.Amount.describe())
print(cl('-----------------------------------------',
attrs = ['bold']))
print(cl('FRAUD CASE AMOUNT STATS', attrs = ['bold']))
print(fraud_cases.Amount.describe())
print(cl('-----------------------------------------',
attrs = ['bold']))
```

```
------------------------------
CASE AMOUNT STATISTICS
------------------------------
NON-FRAUD CASE AMOUNT STATS
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
------------------------------
FRAUD CASE AMOUNT STATS
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
------------------------------
```

Machine Learning: Train and Test Data Split

After computing an exploratory data analysis to familiarize ourselves with the data and understand its strengths and weaknesses, machine learning can occur. The code printed below is the train and test data split. The train and test data split will use a 80% to 20% ratio of the data that is randomly selected. It is important to randomly select the data to reduce bias and error. The training data set allows for the model to learn - and later teach itself - how to predict and interpret the data. Under these circumstances, it allows the system to interpret this data set to predict and prevent future fraudulent cases. The testing data set is the other portion of the data that is used to test the performance of a system. The training data set is primarily used after the testing data set to diminish the chance of error.

Data Split

```
X = df.drop('Class', axis = 1).values
y = df['Class'].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.2, random_state = 0)
```

Print the

```
print(cl('X_train samples : ', attrs = ['bold']), X_train[:1])
```

Data Split

```
print(cl('X_test samples : ', attrs = ['bold']), X_test[0:1])
print(cl('y_train samples : ', attrs = ['bold']),
y_train[0:10])
print(cl('y_test samples : ', attrs = ['bold']), y_test[0:10])
```

```
----------------------------------------
X_train samples :  [[-1.11504743e+00  1.03558276e+00  8.00712441e-01 -1.06039825e+00
   3.26211690e-02  8.53422160e-01 -6.14243480e-01 -3.23116112e+00
   1.53994798e+00 -8.16908791e-01 -1.30559201e+00  1.08177199e-01
  -8.59609580e-01 -7.19342108e-02  9.06655628e-01 -1.72092961e+00
   7.97853221e-01 -6.75939779e-03  1.95677806e+00 -6.44895565e-01
   3.02038533e+00 -5.39617976e-01  3.31564886e-02 -7.74945766e-01
   1.05867812e-01 -4.30853482e-01  2.29736936e-01 -7.05913036e-02
   1.29500000e+01]]
X_test samples :  [[-3.23333572e-01  1.05745525e+00 -4.83411518e-02 -6.07204308e-01
   1.25982115e+00 -9.17607168e-02  1.15910150e+00 -1.24334606e-01
  -1.74639536e-01 -1.64440065e+00 -1.11886302e+00  2.02647310e-01
   1.14596495e+00 -1.80235956e+00 -2.47177932e-01 -6.09453515e-02
   8.46605738e-01  3.79454387e-01  8.47262245e-01  1.86409421e-01
  -2.07098267e-01 -4.33890272e-01 -2.61613283e-01 -4.66506063e-02
   2.11512300e-01  8.29721214e-03  1.08494430e-01  1.61139167e-01
   4.00000000e+01]]
y_train samples :  [0 0 0 0 0 0 0 0 0 0]
y_test samples :  [0 0 0 0 0 0 0 0 0 0]
```

Machine Learning: Modeling the Data Split

Decision Tree

```
tree_model = DecisionTreeClassifier(max_depth = 4,
criterion = 'entropy')
tree_model.fit(X_train, y_train)
tree_yhat = tree_model.predict(X_test)
```

The Decision Tree algorithm helps sort examples through breaking down more specific individual data within the context of a broader more general sample size. In particular, the Decision Tree model incorporates the use of other algorithms that later decides whether or not to further split the data. The entirety of the data fed to the algorithm (also referred to as the root of the tree) will follow a chain system that answers yes or no questions (called a node) until it reaches the final point (the leaf of the tree) (Andrade). The Decision Tree model uses the training data to make predictions of future data. (Andrade).

K-Nearest Neighbors (KNN)

```
n = 5
knn = KNeighborsClassifier(n_neighbors = n)
knn.fit(X_train, y_train)
knn_yhat = knn.predict(X_test)
```

K-Nearest Neighbors is an algorithm that clusters data with known categories. Next, a new and unknown data point is entered. To calculate the category of this new data point, K-Nearest Neighbors searches for its nearest neighbor to categorize the new data point. In the context of credit card fraud, it will insert new data and categorize it as being fraudulent or non-fraudulent by its nearest neighbor (Joshstarmer).

Logistic Regression

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_yhat = lr.predict(X_test)
```

Most common to binary datasets is the Logistic Regression algorithm. A Logistic Regression algorithm uses labeled input and output training data to predict the outcomes of future binary data. In the context of detecting fraudulent credit card charges, a Logistic Regression algorithm uses the sigmoid function – a logistic function – to find the probability of a fraudulent or non-fraudulent credit card (Andrade).

Support Vector Machine (SVM)

```
svm = SVC()
svm.fit(X_train, y_train)
svm_yhat = svm.predict(X_test)
```

A Support Vector Machine (SVM) uses labeled input and output training data and creates a sensitive threshold midpoint between the range of data points. In doing so, when future data is imported into the support vector machine, it more precisely directs the system to calculate an accurate response (Andrade). Therefore, Support Vector Machines have low bias because they use the training data to calculate a sensitive threshold with a maximum margin (Joshstarmer).

Random Forest Tree

```
rf = RandomForestClassifier(max_depth = 4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)
```

A Random Forest Tree is similar to a Decision Tree algorithm in that it follows a similar logic but produces multiple Decision Trees using randomly selected data, thereby, a Random Forest Tree is a more advanced Decision Tree algorithm (Andrade). Random Forest Tree algorithm selects data randomly through a system called bootstrap. After the data is inserted into the Random Forest Tree, it uses a method called bagging to generate multiple Decision Trees.

XGBoost

```
xgb = XGBClassifier(max_depth = 4)
xgb.fit(X_train, y_train)
xgb_yhat = xgb.predict(X_test)
```

XGBoost uses a gradient boosting framework that uses open-ended, and unstructured data. XGBoost avoids bias through parallel processes and tree-pruning. Parallelization in XGBoost uses an initialization that changes the order of loops that the data is sequenced through. Tree-pruning in XGBoost is done through ‘max_depth’ which starts the data sequencing backwards (Morde). In comparison to

the other algorithms mentioned previously, XGBoost is one of the best because of its shorter training time and prediction power.

Evaluation: Accuracy Score

An accuracy score of the algorithms (Decision Tree, K-Nearest Neighbors, Logistic Regression, Support Vector Machine, Random Forest Tree, XGBoost) is a straightforward method to test how accurate the selected algorithm is at classifying the samples correctly (Mohajon).

Accuracy is calculated using the formula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy Score

```
print(cl('ACCURACY SCORE', attrs = ['bold']))
print(cl('------------------------------------------------------------------------------
', attrs = ['bold']))
print(cl('Accuracy score of the Decision Tree model is
{}'.format(accuracy_score(y_test, tree_yhat)), attrs = ['bold']))
print(cl('------------------------------------------------------------------------------
', attrs = ['bold']))
print(cl('Accuracy score of the KNN model is {}'.format(accuracy_score(y_test,
knn_yhat)), attrs = ['bold'], color = 'green'))
print(cl('------------------------------------------------------------------------------
', attrs = ['bold']))
print(cl('Accuracy score of the Logistic Regression model is
{}'.format(accuracy_score(y_test, lr_yhat)), attrs = ['bold'], color = 'red'))
print(cl('------------------------------------------------------------------------------
', attrs = ['bold']))
print(cl('Accuracy score of the SVM model is {}'.format(accuracy_score(y_test,
svm_yhat)), attrs = ['bold']))
print(cl('------------------------------------------------------------------------------
', attrs = ['bold']))
print(cl('Accuracy score of the Random Forest Tree model is
{}'.format(accuracy_score(y_test, rf_yhat)), attrs = ['bold']))
print(cl('------------------------------------------------------------------------------
', attrs = ['bold']))
print(cl('Accuracy score of the XGBoost model is {}'.format(accuracy_score(y_test,
xgb_yhat)), attrs = ['bold']))
print(cl('------------------------------------------------------------------------------
', attrs = ['bold']))
```

```
ACCURACY SCORE
----------------------------------------------------------------------
Accuracy score of the Decision Tree model is 0.9993679997191109
----------------------------------------------------------------------
Accuracy score of the KNN model is 0.9993328885923949
----------------------------------------------------------------------
Accuracy score of the Logistic Regression model is 0.9991573329588147
----------------------------------------------------------------------
Accuracy score of the SVM model is 0.998735999438222
----------------------------------------------------------------------
Accuracy score of the Random Forest Tree model is 0.9992977774656788
----------------------------------------------------------------------
Accuracy score of the XGBoost model is 0.9994733330992591
----------------------------------------------------------------------
```

Based on the results pictured above, it is evident that all algorithms used on this dataset are accurate. In other words, each algorithm classifies the samples correctly as it has a high accuracy score.

Evaluation: F1 Score

An F1-Score is described as a "harmonic mean of precision and recall" (Sharma). Precision focuses on the prediction of positive cases that were predicted correctly by dividing the true positive cases by the actual results (Sharma). Recall, also referred to as sensitivity, focuses on how many positive cases were predicted correctly by dividing the true positive cases by the predictive results (Sharma).

The F1-Score is calculated using the formula:

$$F\text{-}score = \frac{2 * Recall * Precision}{Recall + Precision}$$

F1 Score

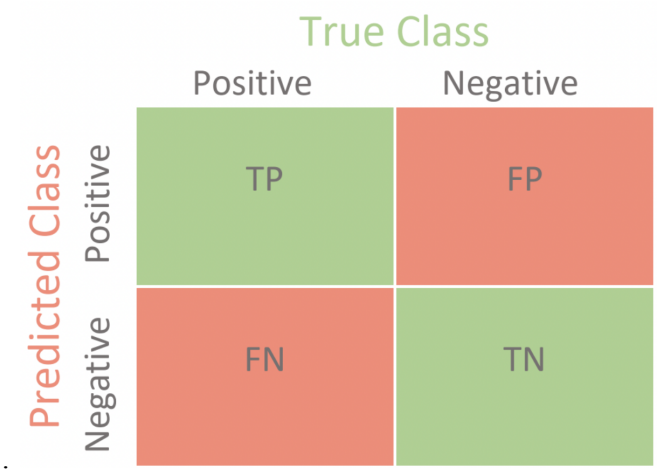
```
print(cl('F1 SCORE', attrs = ['bold']))
print(cl('----------------------------------------------------------------------
', attrs = ['bold']))
print(cl('F1 score of the Decision Tree model is {}'.format(f1_score(y_test,
tree_yhat)), attrs = ['bold']))
print(cl('----------------------------------------------------------------------
', attrs = ['bold']))
print(cl('F1 score of the KNN model is {}'.format(f1_score(y_test, knn_yhat)),
attrs = ['bold'], color = 'green'))
print(cl('----------------------------------------------------------------------
', attrs = ['bold']))
print(cl('F1 score of the Logistic Regression model is {}'.format(f1_score(y_test,
```

```
lr_yhat)), attrs = ['bold'], color = 'red'))
print(cl('------------------------------------------------------------------------
', attrs = ['bold']))
print(cl('F1 score of the SVM model is {}'.format(f1_score(y_test, svm_yhat)),
attrs = ['bold']))
print(cl('------------------------------------------------------------------------
', attrs = ['bold']))
print(cl('F1 score of the Random Forest Tree model is {}'.format(f1_score(y_test,
rf_yhat)), attrs = ['bold']))
print(cl('------------------------------------------------------------------------
', attrs = ['bold']))
print(cl('F1 score of the XGBoost model is {}'.format(f1_score(y_test, xgb_yhat)),
attrs = ['bold']))
print(cl('------------------------------------------------------------------------
', attrs = ['bold']))
```

```
F1 SCORE
------------------------------------------------------------------------
F1 score of the Decision Tree model is 0.8105263157894738
------------------------------------------------------------------------
F1 score of the KNN model is 0.7865168539325842
------------------------------------------------------------------------
F1 score of the Logistic Regression model is 0.7176470588235294
------------------------------------------------------------------------
F1 score of the SVM model is 0.5
------------------------------------------------------------------------
F1 score of the Random Forest Tree model is 0.7727272727272727
------------------------------------------------------------------------
F1 score of the XGBoost model is 0.8421052631578948
------------------------------------------------------------------------
```

Confusion Matrix:

For the next part, a confusion matrix will be computed using different algorithms of the train and test data split. The purpose of computing a confusion matrix for each of the algorithms is to illustrate which algorithm has the best performance. Under these circumstances, the product of this system will be to find the most accurate and efficient algorithm to detect future credit card fraud based on the train and test data split (Mohajon). Illustrated below is the general explanation of a confusion matrix model.



Confusion matrix uses the terms 'positive/negative' and 'true/false' to describe predicted values and actual values. Therefore, predicted values are described by using positive and negative whereas actual values are described by using true and false (Sharma).

'TP,' or True Positive would mean that it was predicted that non-fraudulent cases occurred when non-fraudulent cases did occur. Author Joydwip Mohajon describes 'TP' cases as "the number of predictions where the classifier correctly predicts the positive class as positive." In short, it was predicted to be positive and it was true.

'TN,' or True Negatives would mean that it was predicted that non-fraudulent cases did not occur and non-fraudulent cases did not occur. Author Joydwip Mohajon describes 'TP' cases as "the number of predictions where the classifier correctly predicts the negative class as negative." In short, it was predicted to be negative and it was true.

'FP,' or False Positive would mean that it was predicted that non-fraudulent cases did occur but non-fraudulent cases did not occur. In other words, a False Positive is a Type 1 Error. Author Joydwip Mohajon describes 'TP' cases as "the number of predictions where the classifier incorrectly predicts the negative class as positive." In short, it was predicted to be positive and it was false

'FN,' or False Negatives would mean that it was predicted that non-fraudulent cases did not occur but non-fraudulent cases did not actually occur. In other words, a False Negative is a Type II Error. Author Joydwip Mohajon describes 'TP' cases as "the number of predictions where the classifier incorrectly predicts the positive class as negative." In short, it was predicted to be negative and it was false.

Defining the Plot Function

```
def plot_confusion_matrix(cm, classes, title, normalize = False, cmap =
plt.cm.Blues):
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
```

```
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Compute Confusion Matrix for the Models:

Next, we have to compute the confusion matrix for the models to visualize the performance of the given algorithms.

Decision Tree	`tree_matrix = confusion_matrix(y_test, tree_yhat, labels = [0, 1])`
K-Nearest Neighbors	`knn_matrix = confusion_matrix(y_test, knn_yhat, labels = [0, 1])`
Logistic Regression	`lr_matrix = confusion_matrix(y_test, lr_yhat, labels = [0, 1])`
Support Vector Machine (SVM)	`svm_matrix = confusion_matrix(y_test, svm_yhat, labels = [0, 1])`
Random Forest Tree	`rf_matrix = confusion_matrix(y_test, rf_yhat, labels = [0, 1])`
XGBoost	`xgb_matrix = confusion_matrix(y_test, xgb_yhat, labels = [0, 1])`

Plot the Confusion Matrix

Plot the Confusion Matrix	`plt.rcParams['figure.figsize'] = (6, 6)`

How to Interpret the Confusion Matrix Graph for Each Algorithm

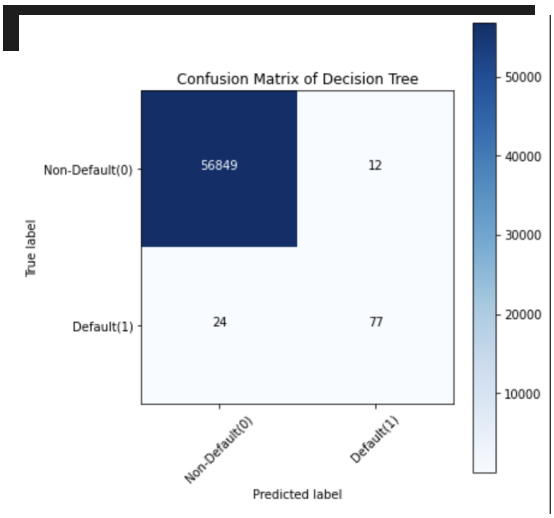
Non-fraudulent	it was predicted that non-fraudulent cases occurred when non-fraudulent cases did occur.	Incorrectly identified a credit card transaction that was fraudulent as a non-fraudulent charge
Fraudulent	Incorrectly identified cases as fraudulent when the credit card transactions were not fraudulent	it was predicted that fraudulent cases did occur when fraudulent cases actually occurred
	Non-fraudulent	Fraudulent

Decision Tree

```
tree_cm_plot =
plot_confusion_matrix(tree_matrix,

classes =
['Non-Default(0)','Default(1)'],

normalize = False, title =
'Decision Tree')
plt.savefig('tree_cm_plot.png')
plt.show()
```



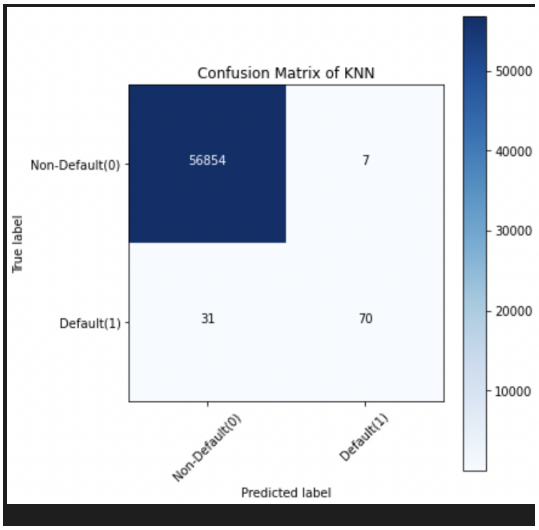
The confusion matrix of the Decision Tree algorithm illustrates that 56,849 predicted values were positive and true. In other words, the predicted values were positive and the actual values were true. The Decision Tree algorithm classified 77 cases as fraudulent. Although, this algorithm incorrectly identified 24 cases as fraudulent when the credit card transactions were not. Moreover, it incorrectly identified a credit card transaction that was fraudulent as a non-fraudulent charge 12 times.

K-Nearest Neighbors

```
knn_cm_plot =
plot_confusion_matrix(knn_matrix,

classes =
['Non-Default(0)','Default(1)'],

normalize = False, title = 'KNN')
plt.savefig('knn_cm_plot.png')
plt.show()
```



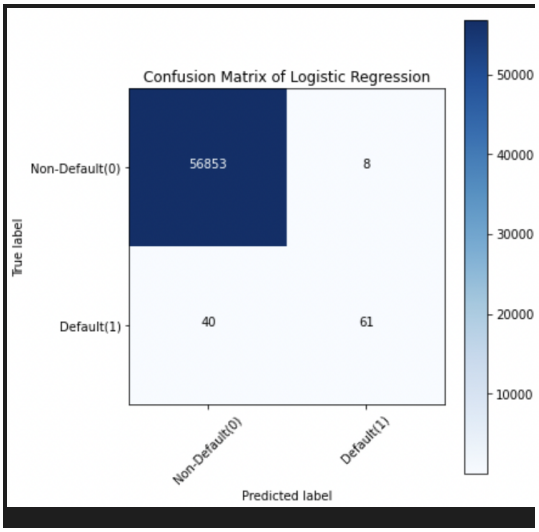
The confusion matrix of the K-Nearest Neighbors algorithm illustrates that 56,854 predicted values were positive and true. In other words, the predicted values were positive and the actual values were true. The K-Nearest Neighbors algorithm classified 70 cases as fraudulent. Although, this algorithm incorrectly identified 31 cases as fraudulent when the credit card transactions were not. Moreover, it incorrectly identified a credit card transaction that was fraudulent as a non-fraudulent charge 7 times.

Logistic Regression

```
lr_cm_plot =
plot_confusion_matrix(lr_matrix,

classes =
['Non-Default(0)','Default(1)'],

normalize = False, title =
'Logistic Regression')
plt.savefig('lr_cm_plot.png')
plt.show()
```



The confusion matrix of the Logistic Regression algorithm illustrates that 56,853

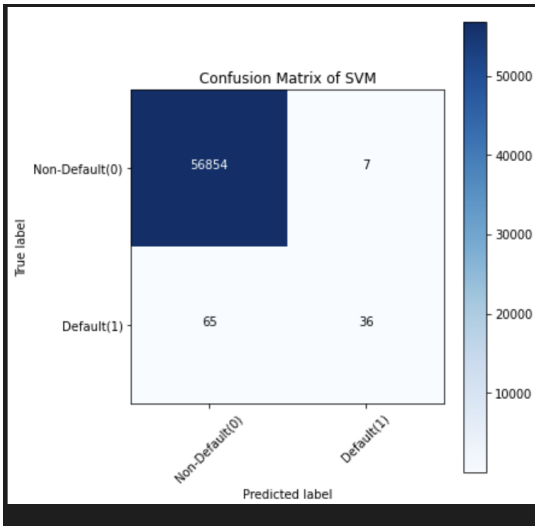
predicted values were positive and true. In other words, the predicted values were positive and the actual values were true. The Logistic Regression algorithm classified 61 cases as fraudulent. Although, this algorithm incorrectly identified 40 cases as fraudulent when the credit card transactions were not. Moreover, it incorrectly identified a credit card transaction that was fraudulent as a non-fraudulent charge 8 times.

Support Vector Machine (SVM)

```
svm_cm_plot =
plot_confusion_matrix(svm_matrix,

classes =
['Non-Default(0)','Default(1)'],

normalize = False, title = 'SVM')
plt.savefig('svm_cm_plot.png')
plt.show()
```



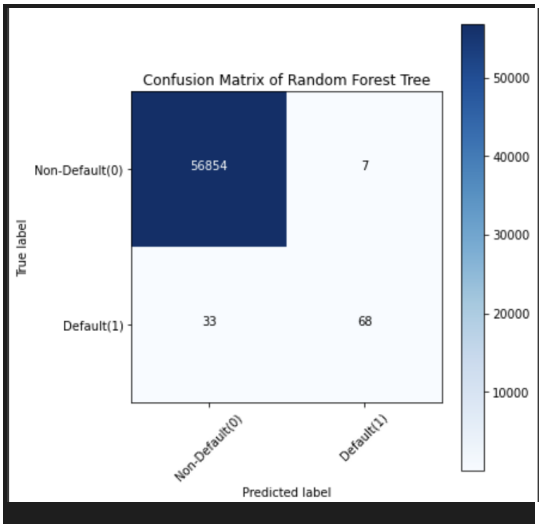
The confusion matrix of the Support Vector Machine (SVM) algorithm illustrates that 56,854 predicted values were positive and true. In other words, the predicted values were positive and the actual values were true. The Support Vector Machine algorithm classified 36 cases as fraudulent. Although, this algorithm incorrectly identified 65 cases as fraudulent when the credit card transactions were not. Moreover, it incorrectly identified a credit card transaction that was fraudulent as a non-fraudulent charge 7 times.

Random Forest Tree

```
rf_cm_plot =
plot_confusion_matrix(rf_matrix,

classes =
['Non-Default(0)','Default(1)'],

normalize = False, title = 'Random
Forest Tree')
plt.savefig('rf_cm_plot.png')
plt.show()
```



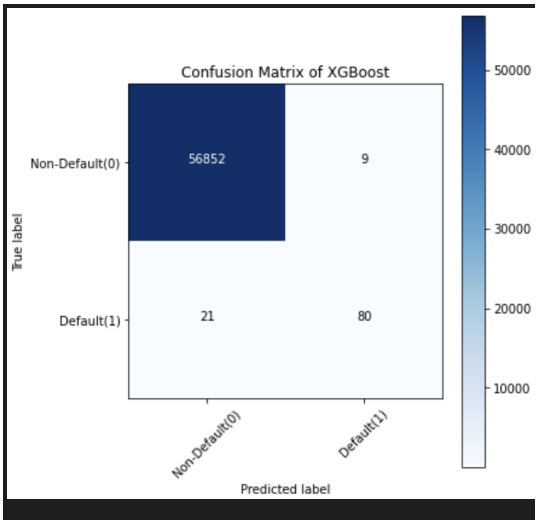
The confusion matrix of the Random Forest Tree algorithm illustrates that 56,854 predicted values were positive and true. In other words, the predicted values were positive and the actual values were true. The Random Forest Tree algorithm classified 68 cases as fraudulent. Although, this algorithm incorrectly identified 33 cases as fraudulent when the credit card transactions were not. Moreover, it incorrectly identified a credit card transaction that was fraudulent as a non-fraudulent charge 7 times.

XGBoost

```
xgb_cm_plot =
plot_confusion_matrix(xgb_matrix,

classes =
['Non-Default(0)','Default(1)'],

normalize = False, title =
'XGBoost')
plt.savefig('xgb_cm_plot.png')
plt.show()
```



The confusion matrix of the XGBoost algorithm illustrates that 56,852 predicted values were positive and true. In other

words, the predicted values were positive and the actual values were true. The XGBoost algorithm classified 80 cases as fraudulent. Although, this algorithm incorrectly identified 21 cases as fraudulent when the credit card transactions were not. Moreover, it incorrectly identified a credit card transaction that was fraudulent as a non-fraudulent charge 9 times.

Conclusion

As mentioned previously, the XGBoost algorithm in machine learning is one of the better algorithms for predicting suspicious activity and fraudulent credit card transactions because it uses tree-pruning and parallelization. The confusion matrix of the XGBoost found that 80 cases were fraudulent and only incorrectly identified a credit card transaction that was fraudulent as a non-fraudulent charge 9 times. It is clear that the XGBoost algorithm is the better algorithm to use to detect fraud.

Works Cited

Andrade, Frank. "6 Machine Learning Algorithms Anyone Learning Data Science Should Know." *Medium*, Towards Data Science, 15 June 2022, https://towardsdatascience.com/6-machine-learning-algorithms-anyone-learning-data-science-should-know-cb6c388a6fb3.

"Assistance for Victims of Compromised Credit Card Information." *FBI*, FBI, 19 Dec. 2014, https://www.fbi.gov/resources/victim-services/seeking-victim-information/assistance-for-victims-of-compromised-credit-card-information.

Boitnott, John. "Credit Card Fraud Statistics." *Self Financial*, Self Financial, 11 Nov. 2020, https://www.self.inc/info/credit-card-fraud-statistics/.

"Decision Tree Algorithm, Explained." *KDnuggets*, https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html.

joshstarmer. "StatQuest: K-Nearest Neighbors, Clearly Explained." *YouTube*, YouTube, 26 June 2017, https://www.youtube.com/watch?v=HVXime0nQeI.

joshstarmer. "Support Vector Machines Part 1 (of 3): Main Ideas!!!" *YouTube*, YouTube, 30 Sept. 2019, https://www.youtube.com/watch?v=efR1C6CvhmE.

Mullen, Caitlin. "Card Industry Faces $400B in Fraud Losses over next Decade, Nilson Says." *Payments Dive*, 14 Dec. 2021, www.paymentsdive.com/news/card-industry-faces-400b-in-fraud-losses-over-next-decade-nilson-says/611521.

Mohajon, Joydwip. "Confusion Matrix for Your Multi-Class Machine Learning Model." *Medium*, Towards Data Science, 24 July 2021, https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826.

Morde, Vishal. "XGBoost Algorithm: Long May She Reign!" *Medium*, Towards Data Science, 8 Apr. 2019, https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d.

Savasta, Mirko. "PCA: A Linear Transformation." *Medium*, Analytics Vidhya, 26 Oct. 2021, https://medium.com/analytics-vidhya/pca-a-linear-transformation-f8aacd4eb007.

Savasta, Mirko. "PCA: A Linear Transformation." *Medium*, Analytics Vidhya, 26 Oct. 2021, https://medium.com/analytics-vidhya/pca-a-linear-transformation-f8aacd4eb007.

Sharma, Prateek. "Decoding the Confusion Matrix." *Medium*, Towards Data Science, 9 Mar. 2022, https://towardsdatascience.com/decoding-the-confusion-matrix-bb4801decbb.

Sridhar, Aditya. “A Beginner's Guide to Credit Card Fraud Detection Using Machine Learning.” *Medium*, Medium, 11 Jan. 2022, https://medium.com/@adityas03/a-beginners-guide-to-credit-card-fraud-detection-using-machine-learning-bee556426951.

ULB, Machine Learning Group -. “Credit Card Fraud Detection.” *Kaggle*, 23 Mar. 2018, https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.