Credit Card Fraud Detection Using Machine Learning & Python

Final Project Report

Submitted by:

Adesh Padwal - app27@njit.edu
Daanish Quadri - dq33@njit.edu
Sachin Singh Satyanaraya - ss4725@njit.edu

Table of Contents

1. Introduction

Fraud in credit card transactions refers to the illegal and undesired use of a credit card account by someone other than the account owner. It is possible to halt this misuse with the necessary preventative measures, and it is also possible to study the behavior of such fraudulent operations to reduce future occurrences and safeguard against them. To put it another way, credit card fraud may be described as an instance where someone uses another person's credit card without the owner's or the card's issuing authority being aware of it. Monitoring user populations' behavior is a key component of fraud detection because it allows fraud, intrusion, and defaulting to be identified as well as other objectionable behavior. This is a pertinent issue that must be addressed by fields like machine learning and data science, where an automated solution is possible. From the standpoint of learning, this issue is particularly difficult since it is characterized by many characteristics, such class imbalance. There are much more legitimate transactions than fraudulent ones. Additionally, the statistical characteristics of the transaction patterns frequently vary over time.

It is important that credit card companies can recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. Here, comes the need for a system that can track the pattern of all the transactions and if any pattern is abnormal then the transaction should be aborted. In this project, We are trying to determine which model is best to detect fraudulent activity.

We have many machine learning algorithms that can help us classify abnormal transactions. The only requirement is the past data and the suitable algorithm that can fit our data in a better form.

2. Importing the libraries

We use python as the programming language and the libraries we import for this project are as follows:

```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
```

 # text customization

```
from termcolor import colored as cl
```

#Packages related to data visualization

```
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.impute import MissingIndicator, SimpleImputer
from sklearn.preprocessing import  PolynomialFeatures, KBinsDiscretizer, FunctionTransformer
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,LabelBinarizer,
OrdinalEncoder
import statsmodels.formula.api as smf
import statsmodels.tsa as tsa
from sklearn.linear_model import LogisticRegression, LinearRegression, ElasticNet, Lasso, Ridge
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.tree import *
from sklearn.ensemble import BaggingClassifier, BaggingRegressor, RandomForestClassifier,
RandomForestRegressor
from sklearn.ensemble import GradientBoostingClassifier,GradientBoostingRegressor,
AdaBoostClassifier, AdaBoostRegressor
from sklearn.svm import LinearSVC, LinearSVR, SVC, SVR
from xgboost import XGBClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

3. Loading the Dataset

We have taken the dataset for Credit Card Fraud Detection from the link below.

https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

4. Splitting the Dataset

Before splitting the data into train & test. we defined dependent and independent variables.
The dependent variable is known as X and the independent variable is known as y.
Then, we have split the dataset into training and testing datasets.

5. Exploratory Data Analysis (EDA)

Dataset contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we could not be able to get the original features and more background information about the data. Features (V1-V28) are the principal components obtained with PCA, the only features which have not been transformed with PCA are '*Time*' and '*Amount*'.

Features:

1. Time: contains the seconds elapsed between each transaction and the first transaction in the dataset.
2. Amount: feature 'Amount' is the transaction Amount. this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable, and it takes value 1 in case of fraud and 0 otherwise.

6. Code:

```
import os
import warnings
warnings.filterwarnings('ignore')
# Packages related to data importing, manipulation, exploratory data analysis and data understanding
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
# text customization package
from termcolor import colored as cl
# Packages related to data visualizaiton
import seaborn as sns
import matplotlib.pyplot as plt

# Setting plot sizes and type of plot
plt.rc("font", size=14)
plt.rcParams['axes.grid'] = True
plt.figure(figsize=(6,3))
plt.gray()
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

from sklearn import metrics
from sklearn.impute import MissingIndicator, SimpleImputer
from sklearn.preprocessing import  PolynomialFeatures, KBinsDiscretizer, FunctionTransformer
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, LabelBinarizer, OrdinalEncoder
import statsmodels.formula.api as smf
import statsmodels.tsa as tsa
from sklearn.linear_model import LogisticRegression, LinearRegression, ElasticNet, Lasso, Ridge
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.tree import *
from sklearn.ensemble import BaggingClassifier,
BaggingRegressor,RandomForestClassifier,RandomForestRegressor
from sklearn.ensemble import GradientBoostingClassifier,GradientBoostingRegressor, AdaBoostClassifier,
AdaBoostRegressor
from sklearn.svm import LinearSVC, LinearSVR, SVC, SVR
from xgboost import XGBClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

```
# importing data from csv file
data=pd.read_csv("creditcard.csv")

# distribution of transaction
Total_transactions = len(data)
normal = len(data[data.Class == 0])
fraudulent = len(data[data.Class == 1])
fraud_percentage = round(fraudulent/normal*100, 2)
print(cl('Total number of Trnsactions are {}'.format(Total_transactions), attrs = ['bold']))
print(cl('Number of Normal Transactions are {}'.format(normal), attrs = ['bold']))
print(cl('Number of fraudulent Transactions are {}'.format(fraudulent), attrs = ['bold']))
print(cl('Percentage of fraud Transactions is {}'.format(fraud_percentage), attrs = ['bold']))

data.info()

print('min and max of data amount ',min(data.Amount),max(data.Amount))

# using standard scaler
sc = StandardScaler()
amount = data['Amount'].values
data['Amount'] = sc.fit_transform(amount.reshape(-1, 1))

print('data shape ',data.shape)
print('dropping external deciding factor (Time)')

# dropping external deciding factor
data.drop(['Time'], axis=1, inplace=True)
print('data shape ',data.shape)
print('dropping duplicate transactions ')

# removing duplicates
data.drop_duplicates(inplace=True)
print('data shape ',data.shape)

#===========================================================================
#============================== Train and Test Split =====================

# dependent and independent variables
X = data.drop('Class', axis = 1).values
y = data['Class'].values

#  splitting the Train and Test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1)

# print(data.shape)
#=================================================================================
```

```
#============================ Model Building ==========================
# 1. Decision Tree

DT = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
DT.fit(X_train, y_train)
dt_yhat = DT.predict(X_test)

print('Accuracy score of the Decision Tree model is {}'.format(accuracy_score(y_test, dt_yhat)))

# checking F1 score
print('F1 score of the Decision Tree model is {}'.format(f1_score(y_test, dt_yhat)))

# checking decision matrix
confusion_matrix(y_test, dt_yhat, labels = [0, 1])
print('Confusion Matrix of the Decision Tree model ')
print(confusion_matrix(y_test, dt_yhat, labels = [0, 1]))

#=========================================================================

# 2. K-Nearest Neighbours

n = 7
KNN = KNeighborsClassifier(n_neighbors = n)
KNN.fit(X_train, y_train)
knn_yhat = KNN.predict(X_test)

# checking accuracy
print('Accuracy score of the K-Nearest Neighbors model is {}'.format(accuracy_score(y_test, knn_yhat)))

# checking F1 score
print('F1 score of the K-Nearest Neighbors model is {}'.format(f1_score(y_test, knn_yhat)))

# checking confusion matrix
confusion_matrix(y_test, knn_yhat, labels = [0, 1])
print('Confusion Matrix of the  K-Nearest Neighbours ')
print(confusion_matrix(y_test, knn_yhat, labels = [0, 1]))


#=========================================================================

# 3. Logistic Regression

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

# checking accuracy
print('Accuracy score of the Logistic Regression model is {}'.format(accuracy_score(y_test, lr_yhat)))
```

```
# checking F1 score
print('F1 score of the Logistic Regression model is {}'.format(f1_score(y_test, lr_yhat)))


# checking confusion matrix
confusion_matrix(y_test, lr_yhat, labels = [0, 1])
print('Confusion Matrix of the Logistic Regression ')
print(confusion_matrix(y_test, lr_yhat, labels = [0, 1]))
#===========================================================================

# 5. Random Forest

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

# checking accuracy
print('Accuracy score of the Random Forest model is {}'.format(accuracy_score(y_test, rf_yhat)))

# checking F1 score
print('F1 score of the Random Forest model is {}'.format(f1_score(y_test, rf_yhat)))
# checking confusion matrix
confusion_matrix(y_test, rf_yhat, labels = [0, 1])
print('Confusion Matrix of the Random Forest ')
print(confusion_matrix(y_test, rf_yhat, labels = [0, 1]))


#==========================================================================

# 6. XGBoost model

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

# checking accuracy
print('Accuracy score of the XGBoost model is {}'.format(accuracy_score(y_test, xgb_yhat)))

# checking F1 score
print('F1 score of the XGBoost model is {}'.format(f1_score(y_test, xgb_yhat)))
# checking confusion matrix
confusion_matrix(y_test, xgb_yhat, labels = [0, 1])
print('Confusion Matrix of the XGBoost model ')
print(confusion_matrix(y_test, xgb_yhat, labels = [0, 1]))
```

7. Results:

```
(base) adeshpadwal@123 Credit-Card-Fraud-Detection-Using-Machine-Learning-Python %  cd /Users/adeshpadwal/Deskt
dwal/.vscode/extensions/ms-python.python-2022.18.2/pythonFiles/lib/python/debugpy/adapter/../../debugpy/launche
Total number of Trnsactions are 284807
Number of Normal Transactions are 284315
Number of fraudulent Transactions are 492
Percentage of fraud Transactions is 0.17
```

From 284807 transactions there are 284315 transactions are normal and 492 transactions are fraudulent which is 0.17%.

Data information:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Minimum and Maximum of Data Amount:

```
min and max of data amount  0.0 25691.16
```

Data Shape before and after dropping external deciding factor (Time) and removing duplicate transactions:

```
data shape  (284807, 31)
dropping external deciding factor (Time)
data shape  (284807, 30)
dropping duplicate transactions
data shape  (275663, 30)
```

Decision Tree Model:

```
Accuracy score of the Decision Tree model is 0.9991583957281328
F1 score of the Decision Tree model is 0.7542372881355933
Confusion Matrix of the Decision Tree model
[[68769    19]
 [   39    89]]
```

K-Nearest Neighbors Model:

```
Accuracy score of the K-Nearest Neighbors model is 0.999288989494457
F1 score of the K-Nearest Neighbors model is 0.7949790794979079
Confusion Matrix of the  K-Nearest Neighbours
[[68772    16]
 [   33    95]]
```

Logistic Regression Model:

```
Accuracy score of the Logistic Regression model is 0.9989552498694062
F1 score of the Logistic Regression model is 0.6666666666666666
Confusion Matrix of the Logistic Regression
[[68772    16]
 [   56    72]]
```

Random Forest Model:

```
Accuracy score of the Random Forest model is 0.9991729061466132
F1 score of the Random Forest model is 0.7397260273972602
Confusion Matrix of the Random Forest
[[68778    10]
 [   47    81]]
```

XG Boost Model:

```
Accuracy score of the XGBoost model is 0.999506645771664
F1 score of the XGBoost model is 0.8495575221238937
Confusion Matrix of the XGBoost model
[[68786     2]
 [   32    96]]
```

Models	Accuracy	F1 Score
Decision Tree	0.999288989494457	0.776255707762557
K-Nearest Neighbors	0.999506645771664	0.8365384615384616
Logistic Regression model	0.9991148644726914	0.6934673366834171
Support vector machine	0.9993615415868594	0.7777777777777779
Random Forest	0.9993615415868594	0.7843137254901961
XG Boost	0.9995211561901445	0.8421052631578947

8. Conclusion:

We have used XGBoost model, K-Nearest Neighbors model and Logistic Regression model, Decision Tree model, Support Vector model and Random Forest model. We have already received **99.95% accuracy** in our credit card fraud detection by using **XGBoost model,** which is highest accuracy we have received among all the six models. This number should not be surprising as our data was balanced towards one class. The good thing that we have noticed from the confusion matrix is that our model is not overfitted.
Based on class imbalance ratio, we decided measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC) because confusion matrix accuracy is not meaningful for unbalanced classification.
Finally, based on our accuracy score, XGBoost gave better accuracy than other two models. The only catch here is the data that we have received for model training is transformed version of PCA.