Credit Card Fraud Detection Using Machine Learning & Python

As we are moving towards the digital world — cybersecurity is becoming a crucial part of our life. When we talk about security in digital life then the main challenge is to find the abnormal activity.

When we make any transaction while purchasing any product online — a good amount of people prefers credit cards. The credit limit in credit cards sometimes helps us me making purchases even if we don't have the amount at that time. but, on the other hand, these features are misused by cyber attackers.

To tackle this problem, we need a system that can abort the transaction if it finds fishy.

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.

Today, 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.

In this article, I will help you in the complete end-to-end model training process — finally, you will get the best model that can classify the transaction into normal and abnormal types.

About the data

The data for this article can be found here. This dataset contains the real bank transactions made by European cardholders in the year 2013. As a security concern, the actual variables are not being shared but — they have been transformed versions of PCA. As a result, we can find 29 feature columns and 1 final class column.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.23279
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.75013
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.41526
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.37320
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.06973

10 rows × 31 columns

Data Snapshot

Importing Necessary Libraries

It is a good practice to import all the necessary libraries in one place — so that we can modify them quickly.

For this credit card data, the features that we have in the dataset are the transformed version of PCA, so we will not need to perform the feature selection again. Otherwise, it is recommended to use RFE, RFECV, SelectKBest and VIF score to find the best features for your model.

```
#Packages related to general operating system & warning

import os
import warnings
warnings.filterwarnings('ignore')
#Packages related to data importing, manipulation, exploratory data #analysis, data understanding
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
from termcolor import colored as cl # text customization
#Packages related to data visualizaiton
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
#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 train_test_split, GridSearchCV
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 DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz, export
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 Dataset

Importing the dataset is pretty much simple. You can use pandas module in python to import it.

Run the below command to import your data.

```
data=pd.read_csv("creditcard.csv")
```

Data Processing & Understanding

The one main thing you will notice about this data is that — the dataset is imbalanced towards a feature. Which seems valid for such kind of data. Because today many banks have adopted different security mechanisms — so it is harder for hackers to make such moves.

Still, sometimes when there is some vulnerability in the system — the chance of such activities can increase.

That's why we can see the majority of transactions belongs to our datasets are normal and only a few percentages of transactions are fraudulent.

Let's check the transaction distribution.

```
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']))
```

```
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
```

Only 0.17% of transactions are fraudulent.

We can also check for null values using the following line of code.

`data.info()`

```
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)
```

As per the count per column, we have no null values. Also, feature selection is not the case for this use case.

Anyway, you can try applying feature selection mechanisms to check if the results are optimized.

I have observed in our data 28 features are transformed versions of PCA but the Amount is the original one. And, while checking the minimum and maximum is in the amount — I found the difference is huge that can deviate our result.

```
min(data.Amount),max(data.Amount)
```

```
(0.0, 25691.16)
```

In this case, it is a good practice to scale this variable. We can use a standard scaler to make it fix.

```
sc = StandardScaler()
amount = data['Amount'].values
data['Amount'] = sc.fit_transform(amount.reshape(-1, 1))
```

We have one more variable which is the time which can be an external deciding factor — but in our modelling process, we can drop it.

```
data.drop(['Time'], axis=1, inplace=True)
```

We can also check for any duplicate transactions. Before removing any duplicate transaction, we are having 284807 transactions in our data. Let's remove the duplicate and observe the changes.

```
data.shape
```

```
(284807, 30)
```

Run the below line of code to remove any duplicates.

```
data.drop_duplicates(inplace=True)
```

Let's now check the count again.

```
data.shape
```

```
(275663, 30)
```

So, we were having around ~9000 duplicate transactions.

Here we go!! We now have properly scaled data with no duplicate, no missing. Let’s now split it for our model building.

Train & Test Split

Before splitting train & test — we need to define dependent and independent variables. The dependent variable is also known as **X** and the independent variable is known as **y.**

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

Now, let split our train and test data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1)
```

That’s it. We now have two different data set — Train data we will be used for training our model and the data which is unseen will be used for testing.

Model Building

We will be trying different machine learning models one by one. Defining models are much easier. A single line of code can define our model. And, in the same way, a single line of code can fit the model on our data.

We can also tune these models by selecting different optimized parameters. But, if the accuracy is better even with less parameter tuning then — no need to make it complex.

Decision Tree

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

Let’s check the accuracy of our decision tree model.

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

Accuracy score of the Decision Tree model is 0.999288989494457

Checking F1-Score for the decision tree model.

```
print('F1 score of the Decision Tree model is {}'.format(f1_score(y_test, tree_yhat)))
```

F1 score of the Decision Tree model is 0.776255707762557

Checking the confusion matrix:

```
confusion_matrix(y_test, tree_yhat, labels = [0, 1])
```

```
array([[68782,    18],
       [   31,    85]], dtype=int64)
```

Here, the first row represents positive and the second row represents negative. So, we have 68782 as true positive and 18 are false positive. That says, out of 68782+18=68800, we have 68782 that are successfully classified as a normal transaction and 18 were falsely classified as normal — but they were fraudulent.

Let’s now try different models and check their performance.

K-Nearest Neighbors

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

Let's check the accuracy of our K-Nearest Neighbors model.

print('Accuracy score of the K-Nearest Neighbors model is {}'.format(accuracy_score(y_test, knn_yhat)))**Accuracy score of the K-Nearest Neighbors model is 0.999506645771664**

Checking F1-Score for the K-Nearest Neighbors model.

print('F1 score of the K-Nearest Neighbors model is {}'.format(f1_score(y_test, knn_yhat)))**F1 score of the K-Nearest Neighbors model is 0.8365384615384616**

Logistic Regression

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

Let's check the accuracy of our Logistic Regression model.

print('Accuracy score of the Logistic Regression model is {}'.format(accuracy_score(y_test, lr_yhat)))**Accuracy score of the Logistic Regression model is 0.9991148644726914**

Checking F1-Score for the Logistic Regression model.

print('F1 score of the Logistic Regression model is {}'.format(f1_score(y_test, lr_yhat)))**F1 score of the Logistic Regression model is 0.6934673366834171**

Support Vector Machines

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

Let's check the accuracy of our Support Vector Machines model.

print('Accuracy score of the Support Vector Machines model is {}'.format(accuracy_score(y_test, svm_yhat)))**Accuracy score of the Support Vector Machines model is 0.9993615415868594**

Checking F1-Score for the Support Vector Machines model.

```
print('F1 score of the Support Vector Machines model is {}'.format(f1_score(y_test, svm_yhat)))
```
F1 score of the Support Vector Machines model is 0.7777777777777779

Random Forest

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

Let’s check the accuracy of our Random Forest model.

```
print('Accuracy score of the Random Forest model is {}'.format(accuracy_score(y_test, rf_yhat)))
```
Accuracy score of the Random Forest model is 0.9993615415868594

Checking F1-Score for the Random Forest model.

```
print('F1 score of the Random Forest model is {}'.format(f1_score(y_test, rf_yhat)))
```
F1 score of the Random Forest model is 0.7843137254901961

XGBoost

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

Let’s check the accuracy of our XGBoost model.

```
print('Accuracy score of the XGBoost model is {}'.format(accuracy_score(y_test, xgb_yhat)))
```
Accuracy score of the XGBoost model is 0.9995211561901445

Checking F1-Score for the XGBoost model.

```
print('F1 score of the XGBoost model is {}'.format(f1_score(y_test, xgb_yhat)))
```
F1 score of the XGBoost model is 0.8421052631578947

Conclusion

Well, congratulation!! We just received 99.95% accuracy in our credit card fraud detection. 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.

Finally, based on our accuracy score — **XGBoost** is the winner for our case. The only catch here is the data that we have received for model training. The data features are the transformed version of PCA. If the actual features follow a similar pattern then we are doing great!!