CREDIT CARD FRAUD DETECTION

[Final Report]

Date: 30 July 2021

By

VINEET KUMAR

Table of Contents

Introduction

Credit card fraud is a term used for fraudulent activities done to someone via payment cards like credit/debit cards. When a fraud happens, it basically means that an unauthorized person has gained access to someone's account and then uses that to make payments for his/her own benefit. It is one of the most common crimes that occur in the financial domain. There are various type of frauds techniques that have been developed with the development in security measures as well. With early recognition of such frauds, it becomes easy to work on the solutions or reverting back the transaction, which disturbs the whole goal of the fraud. Moreover, by doing so, we get to know of a certain fraudulent activity going on with one of the accounts.

Project Objective

Recognition of the transaction that may have been caused due to a fraud

Role of Machine Learning

Machine learning have a very high chance of detecting any such fraudulent activities. The reason for this is because of the fact that, the model made in machine learning often works with patterns and behaviour of the data. So, when there is any sort of inconsistency, the model is able to find out the difference in the normal trend and identify the fraudulent transaction. The working here principle of the model is to identify the transaction which has certain attributes that differs from the normal and legitimate transactions.

Data Exploration

Our dataset includes the transactions made by Credit card holders in Europe during September 2013. However, due to confidentiality issues, the dataset only includes numerical values achieved via, PCA transformation I.e., V1 to V28 along with the 'Amount' and 'Time'. The dataset is highly unbalanced and only includes 0.17% of transactions which were fraudulent.

The dataset has been collected and analysed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. Here, the column called 'Class' is our dependent variable and will tell us if the transactions are fraudulent or not.

Class 0 - Transaction occurred is NON-FRAUDULENT.
Class 1 - Transaction occurred is FRAUDULENT.

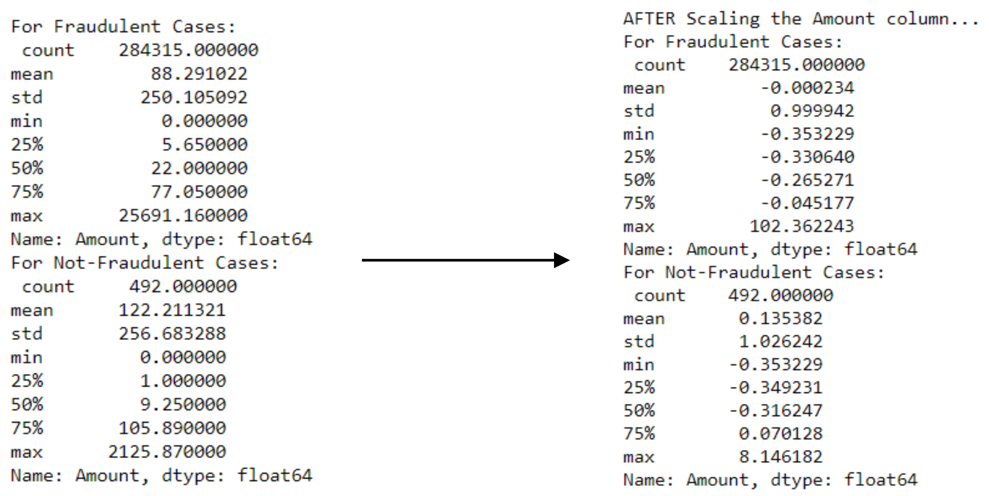
Feature Engineering

The 'Time' feature will not be necessary for building our model, therefore we will start by dropping that feature from our dataset.

Now our next goal is to identify the distribution in our dataset. Since the features have been already created by applying PCA on them, the only other feature that we need to scale is ‘Amount’. As, we can see that the count of fraudulent cases in our dataset is only 492, which is a lot less when compared to the count of non-fraudulent cases, which is 284315. So our data is highly unbalanced. Therefore, this feature needs to be treated carefully.

Now, we will try to scale our data using StandardScaler() method. The reason is to limit most of the values between the range -1 to 1 so that the data points are more compressed. By doing so, now the balance of our data improves and now the difference between our minimum, value and maximum value decreases. This process helps in creating a better model for prediction.

```
In [9]: sc = StandardScaler()
        x['Amount'] = sc.fit_transform(x['Amount'].values.reshape(-1,1))
        print(x['Amount'].head())
```



```
For Fraudulent Cases:
 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
For Not-Fraudulent Cases:
 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
```

```
AFTER Scaling the Amount column...
For Fraudulent Cases:
 count    284315.000000
mean         -0.000234
std           0.999942
min          -0.353229
25%          -0.330640
50%          -0.265271
75%          -0.045177
max         102.362243
Name: Amount, dtype: float64
For Not-Fraudulent Cases:
 count    492.000000
mean       0.135382
std        1.026242
min       -0.353229
25%       -0.349231
50%       -0.316247
75%        0.070128
max        8.146182
Name: Amount, dtype: float64
```

Building training/test samples

Now, once our dataset is ready, we now split it into training and testing parts. We also separate the dependent and independent variables in the dataset. It is to see how the model responds to the unseen data, which will be the testing part.

```
In [11]: X = x.drop('Class', axis = 1).values
         Y = x['Class'].values
```

So here, we have divided our data into:

- Training – 80%
- Testing – 20

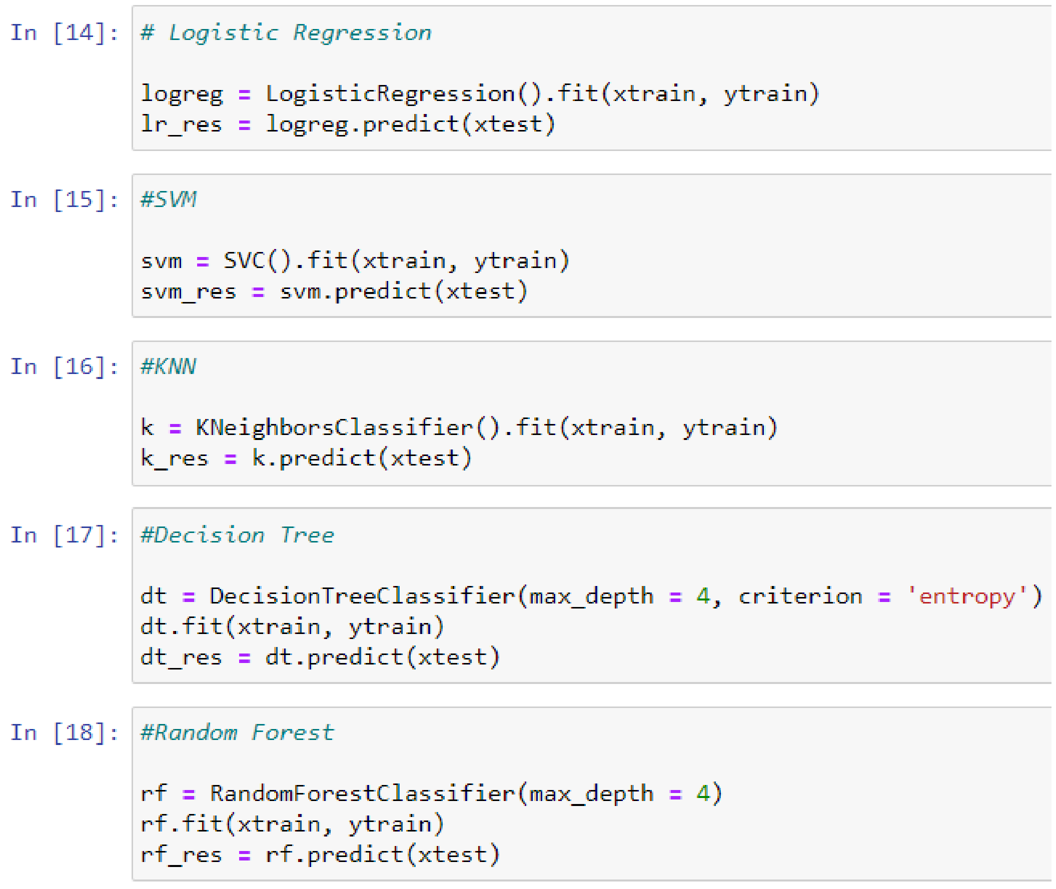
```
In [12]: xtrain, xtest, ytrain, ytest = train_test_split(X,Y, test_size = 0.2)
         print(xtrain, xtest, ytrain, ytest)
```

Model Selection

In this project, we will create 5 different models and then check, accuracy, f1-score for each model and then select the one best suited for this problem. The models that are created are:

- Logistic Regression
- Support Vector Machine
- K-Nearest Neighbors
- Decision Tree
- Random Forest

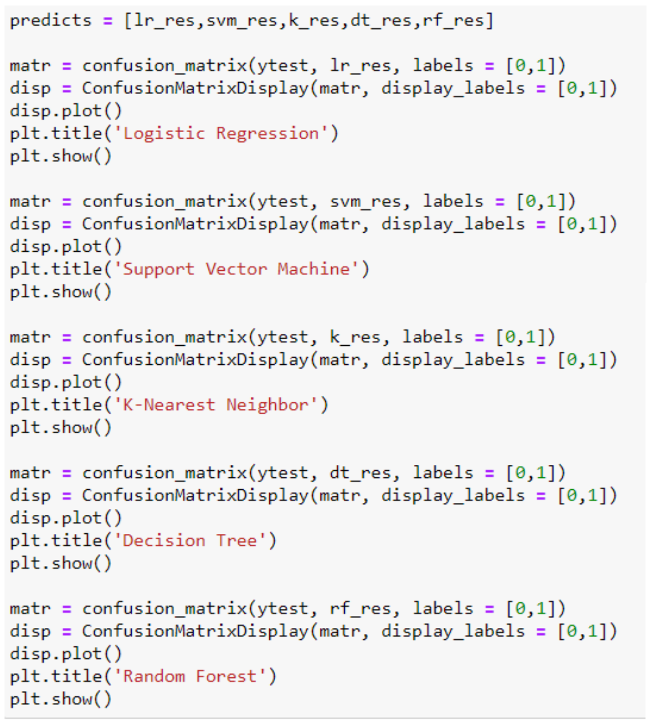
Now, we go ahead and train each of the models with the 80% split of the data.



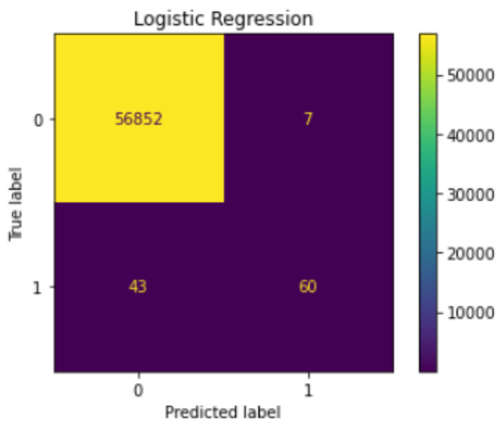
```
In [14]: # Logistic Regression

         logreg = LogisticRegression().fit(xtrain, ytrain)
         lr_res = logreg.predict(xtest)

In [15]: #SVM

         svm = SVC().fit(xtrain, ytrain)
         svm_res = svm.predict(xtest)

In [16]: #KNN

         k = KNeighborsClassifier().fit(xtrain, ytrain)
         k_res = k.predict(xtest)

In [17]: #Decision Tree

         dt = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
         dt.fit(xtrain, ytrain)
         dt_res = dt.predict(xtest)

In [18]: #Random Forest

         rf = RandomForestClassifier(max_depth = 4)
         rf.fit(xtrain, ytrain)
         rf_res = rf.predict(xtest)
```

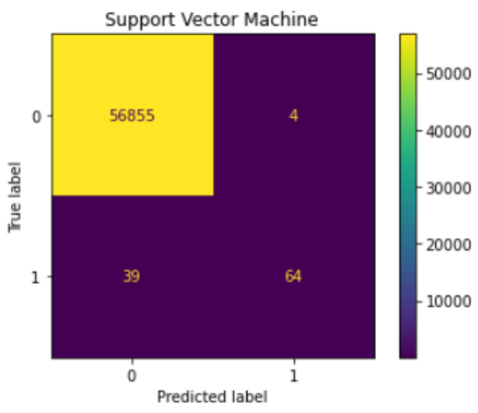
Model Evaluation

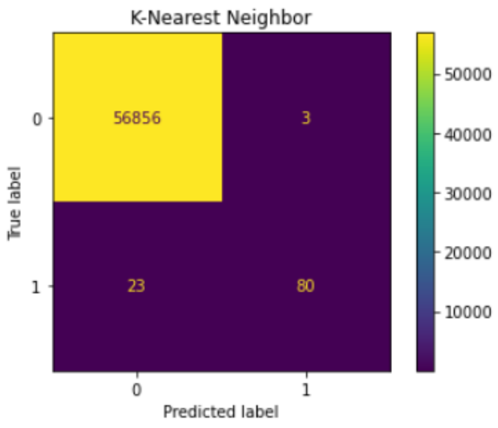
Now once the models have been trained and tested. The next step is to plot a confusion matrix to better visualize the outcome.

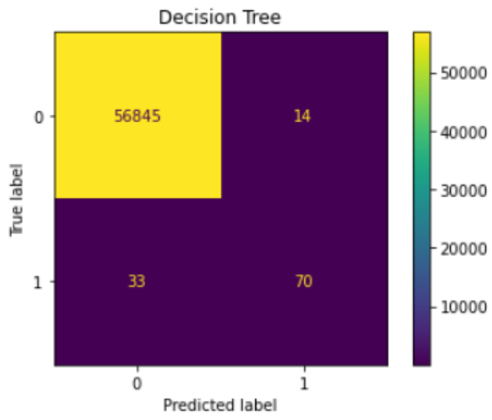


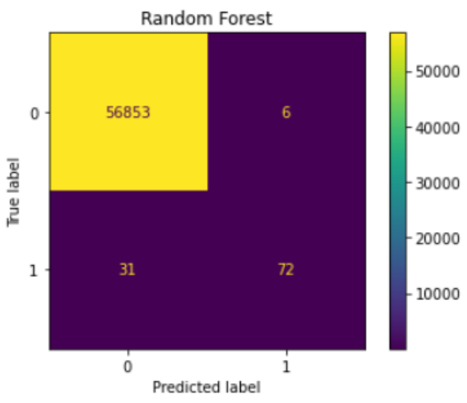
```
predicts = [lr_res,svm_res,k_res,dt_res,rf_res]

matr = confusion_matrix(ytest, lr_res, labels = [0,1])
disp = ConfusionMatrixDisplay(matr, display_labels = [0,1])
disp.plot()
plt.title('Logistic Regression')
plt.show()

matr = confusion_matrix(ytest, svm_res, labels = [0,1])
disp = ConfusionMatrixDisplay(matr, display_labels = [0,1])
disp.plot()
plt.title('Support Vector Machine')
plt.show()

matr = confusion_matrix(ytest, k_res, labels = [0,1])
disp = ConfusionMatrixDisplay(matr, display_labels = [0,1])
disp.plot()
plt.title('K-Nearest Neighbor')
plt.show()

matr = confusion_matrix(ytest, dt_res, labels = [0,1])
disp = ConfusionMatrixDisplay(matr, display_labels = [0,1])
disp.plot()
plt.title('Decision Tree')
plt.show()

matr = confusion_matrix(ytest, rf_res, labels = [0,1])
disp = ConfusionMatrixDisplay(matr, display_labels = [0,1])
disp.plot()
plt.title('Random Forest')
plt.show()
```





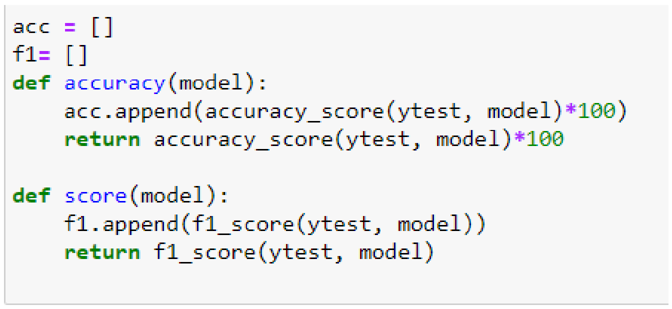






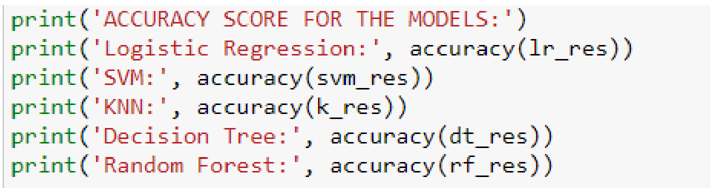
Analysis of Models

Once our models are trained and have predicted their output on the test dataset. We calculate the accuracy and f1 score for the purposes of measuring their performance metrices and find out the best suited model from the selected models for our problem.

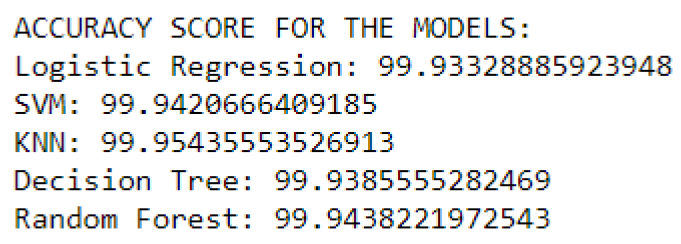


```
acc = []
f1= []
def accuracy(model):
    acc.append(accuracy_score(ytest, model)*100)
    return accuracy_score(ytest, model)*100

def score(model):
    f1.append(f1_score(ytest, model))
    return f1_score(ytest, model)
```

The accuracy measure of a model is calculated as



```
print('ACCURACY SCORE FOR THE MODELS:')
print('Logistic Regression:', accuracy(lr_res))
print('SVM:', accuracy(svm_res))
print('KNN:', accuracy(k_res))
print('Decision Tree:', accuracy(dt_res))
print('Random Forest:', accuracy(rf_res))
```

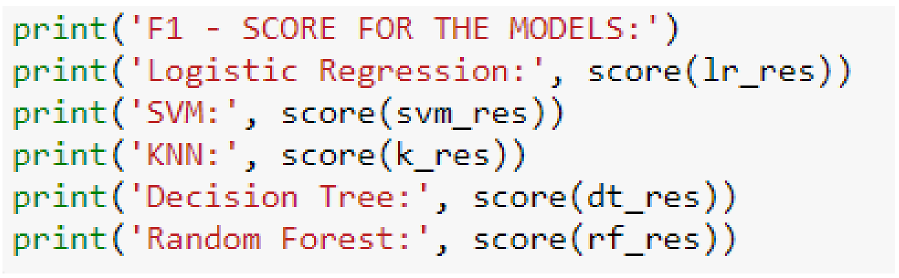


```
ACCURACY SCORE FOR THE MODELS:
Logistic Regression: 99.93328885923948
SVM: 99.9420666409185
KNN: 99.95435553526913
Decision Tree: 99.9385555282469
Random Forest: 99.9438221972543
```

As we can see, that all the models have a accuracy of 99%. This is because since the data is highly unbalanced, it is not a correct measure to be used to judge a model in our situation. Therefore, we will use F1 score.

F1 Score – This is another measure of performance of a model. a good F1 score means that we have low false positives and low false negatives, so we are correctly identifying real threats and we are not disturbed by false alarms.

The F1 score of a model is calculated as



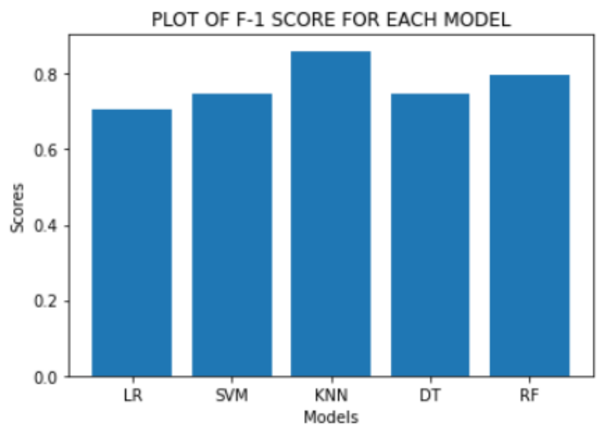
```
print('F1 - SCORE FOR THE MODELS:')
print('Logistic Regression:', score(lr_res))
print('SVM:', score(svm_res))
print('KNN:', score(k_res))
print('Decision Tree:', score(dt_res))
print('Random Forest:', score(rf_res))
```

```
F1 - SCORE FOR THE MODELS:
Logistic Regression: 0.7058823529411764
SVM: 0.7485380116959063
KNN: 0.8602150537634409
Decision Tree: 0.7486631016042781
Random Forest: 0.7955801104972376
```

From the outcome of our performance metrics it can be easily seen that the accuracy is somewhat similar for all the selected models. However, when F1 score is concerned, K-Nearest Neighbours provides the best result with an F1 score of 0.860 The least suited model for this problem is Logistic Regression with an F1 score of 0.705.

```
fscore = np.array(f1)
plt.bar(['LR','SVM','KNN','DT','RF'],fscore)
plt.title('PLOT OF F-1 SCORE FOR EACH MODEL')
plt.xlabel('Models')
plt.ylabel('Scores')
plt.show()
```

To visualize the f-1 scores of the models, we stored it in a list and then converted it into a numpy array and then plot a BAR PLOT.



Key Contributions

The field of data science was born from us, the humans realizing the true worth of the data. Once we did, we are now in an era which is data-driven and this new field is used in almost every field starting from security purposes to financial systems. Machine Learning deals with identifying the hidden patterns behind the data and then pull certain conclusions Then, based on that data, prediction is done. The amount of data that is available today is tremendous and it is not easy to utilize all of it. Therefore, there needs to be proper planning with the datasets being used for prediction and how the original data is manipulated so that the machine is able to understand it.

Future

In this project, we have only used 5 models for classification. However, there are many models which can be used for this problem such as XGBoost and Clustering. Our project works on static dataset but the project can be modified to work upon a live dataset and integrated with a system that can generate an alert whenever a new fraudulent transaction occurs.