

Credit Card Fraud Detection

End Term Report

Name Of the Candidates

Prakash Kumar

Roll Numbers:68

Kolli Kiran Kumar

Roll Numbers:69

Section:KM098



Department of Intelligent Systems

School of Computer Science Engineering

Lovely Professional University, Jalandhar

November-2022

Student Declaration

This is to declare that this report has been written by us. No part of the report is copied from other sources. All information included from other sources have been duly acknowledged. We aver that if any part of the report is found to be copied, we are shall take full responsibility for it.

Prakash Kumar

Signature of Student

Name of Student:

Prakash Kumar

Roll Number: 68

Kiran Kumar

Signature of Student

Name of the Student:

Kolli Kiran Kumar

Roll Number: 69

Place: Lovely Professional University, Phagwara.

Date: 01/11/2022

Table Of Contents

Title	Page No.
1.Introductory description.....	1
1.1 Data Preparation.....	2
1.2 Method 1-Random Forest.....	3
2. challenges involved in credit card fraud detection.....	4
2.1 Loading the data.....	5
2.2 Understanding the data	5
2.3 Describing the data	5
2.4 Imbalance in the data.....	6
2.5 plotting the corelation matrix.....	7
2.6 Training and Testing Data Bifurcation.....	8
2.7 Visualizing the Confusion Matrix.....	9
3. Automating the Machine Learning Pipeline for Credit card fraud detection.....	11
3.1 Loading the dataset.....	12
3.2 Knowing the dataset.....	12
3.3 Code for different models.....	13
3.4 Deploying.....	15

BONAFIDE CERTIFICATE

Certified that this project report “Credit Card Fraud Detection” is the bonafide work of Prakash Kumar and Kolli Kiran Kumar who carried out the project work under my supervision of Dhanpratap Singh.

Signature of the Supervisor

Name of supervisor: Dhanpratap Singh

Academic Designation

ID of Supervisor: 25706

Department of Supervisor:

Computer Science Engineering

Github Link For Credit Card Fraud Detection Project

https://github.com/ShrivastavaPrakash/credit_card_fraud

Machine Learning Techniques for Credit Card Fraud Detection

Online fraud is at an all time high. In our increasingly digital world, it has never been easier for fraudsters to exploit people using the internet. To make matters worse, many consumers have passwords, credit card numbers, and other sensitive information leaked on the dark web. This results in accounts being taken over and new fraudulent lines of credit being opened everyday. Many remain completely unnoticed.

Credit card fraud is among the most common online scams. Credit card numbers, PINs, and security codes can easily be stolen and used to make fraudulent transactions. This can result in huge financial losses for merchants and consumers. However, credit card companies are ultimately responsible to pay back any losses to their customers. Thus, it is extremely important for credit card companies and other financial institutions to be able to detect fraud before it occurs.

Machine learning has become an increasingly accessible and reliable method to detect fraudulent transactions. Using a historical dataset, a machine learning model can be trained to learn patterns behind fraudulent behavior. A model can then be applied to filter out fraudulent transactions and stop them from occurring in real time.

This post will examine 4 commonly used machine learning methods for fraud detection. These include:

- Random Forest
- CatBoost
- Deep Neural Network (DNN)
- Isolation Forest

We will dive into the basics of how to create these models in Python, and compare how they perform against one another. Lets get started!

Data Preparation

To evaluate different machine learning methods for fraud detection, we will use the “Credit Card Fraud Detection” dataset that is publicly available on Kaggle (<https://www.kaggle.com/mlg-ulb/creditcardfraud>). This dataset contains multiple credit card transactions from European card holders in 2013. Many of the features found in this dataset are transformed using PCA, due to the fact that the original dataset contains sensitive personally identifiable information (PII).

Each transaction is truth-marked as either *Fraud* or *Not Fraud* in the column “Class”. Taking a look at class percentages, it is clear that only a tiny percentage of the transactions are fraudulent (00.17 %). This makes training a classifier challenging due to a large class imbalance.

```
#load data
df = pd.read_csv('creditcard.csv')#drop NULL values
df = df.dropna()#drop Time column (contains limited useful information)
df = df.drop("Time", axis = 1)#group data by Class
groups = df.groupby('Class')
fraud = (groups.get_group(1).shape[0] / df.shape[0]) * 100
non_fraud = (groups.get_group(0).shape[0] / df.shape[0]) * 100#print class percentage
print('Percent Fraud: ' + str(fraud) + '%')
print('Percent Not Fraud ' + str(non_fraud) + '%')
```

Method 1: Random Forest

The first method we will use to train a fraud classifier is random forest. A random forest is a popular supervised machine learning algorithm that can be used for both classification and regression tasks. The model works by sampling the training dataset, building multiple decision trees, and then having the output of the decision trees determine a prediction. This model can easily handle large datasets with high dimensionality, and can also handle categorical values easily. However, it can potentially suffer from the large class imbalance in our data.

First, let's initiate a basic random forest model and train it using our training data.

Then, we will retrieve the probabilities of each data point in our test set.

```
#create Random Forest Model
rf = RandomForestClassifier()#fit to training data
rf.fit(X_train, Y_train)#get class probabilities
probabilities = clf.predict_proba(X_test)
y_pred_rf = probabilities[:,1]
```

Next, let's calculate some basic performance metrics. We will calculate false positive rate (FPR), true positive rate (TPR), and the area under the ROC curve.

```
fpr_rf, tpr_rf, thresholds_rf = roc_curve(Y_test, y_pred_rf)
auc_rf = auc(fpr_rf, tpr_rf)
```

Finally, let's plot the performance of our model using a ROC Curve (Receiver Operator Characteristic). This will help us understand the relationship between true positives and false positives that our model produces on our test set. If our model is performing well, we should see a high true positive rate at a low false positive rate.

```
plt.plot(100*fpr_rf, 100*tpr_rf, label= 'Random Forest (area = {:.3f})'.format(auc_rf),
linewidth=2, color = colors[0])plt.xlabel('False positives [%]')
plt.ylabel('True positives [%]')
plt.xlim([0,30])
plt.ylim([60,100])
plt.grid(True)
ax = plt.gca()
ax.set_aspect('equal')
plt.title('Random Forest Model Performance')
plt.legend(loc='best')
```

Looks pretty good! Random Forest appears to be a good machine learning method for fraud detection. It has a consistently high true positive rate (TPR) across varying false positive rates (FPR).

Lets take a look at some more models and see how they compare.

The challenge is to recognize fraudulent credit card transactions so that the customers of credit card companies are not charged for items that they did not purchase.

Main challenges involved in credit card fraud detection are:

1. Enormous Data is processed every day and the model build must be fast enough to respond to the scam in time.
2. Imbalanced Data i.e most of the transactions (99.8%) are not fraudulent which makes it really hard for detecting the fraudulent ones
3. Data availability as the data is mostly private.
4. Misclassified Data can be another major issue, as not every fraudulent transaction is caught and reported.
5. Adaptive techniques used against the model by the scammers.

How to tackle these challenges?

1. The model used must be simple and fast enough to detect the anomaly and classify it as a fraudulent transaction as quickly as possible.
2. Imbalance can be dealt with by properly using some methods which we will talk about in the next paragraph
3. For protecting the privacy of the user the dimensionality of the data can be reduced.
4. A more trustworthy source must be taken which double-check the data, at least for training the model.
5. We can make the model simple and interpretable so that when the scammer adapts to it with just some tweaks we can have a new model up and running to deploy.

Before going to the code it is requested to work on a jupyter notebook. If not installed on your machine you can use [Google colab](#).

You can download the dataset from [this link](#)

If the link is not working please go to [this](#) link and login to kaggle to download the dataset.

Code : Importing all the necessary Libraries

```
# import the necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```



```
import seaborn as sns
from matplotlib import gridspec
```

Code : Loading the Data

```
# Load the dataset from the csv file using pandas
# best way is to mount the drive on colab and
# copy the path for the csv file
data = pd.read_csv("credit.csv")
```

Code : Understanding the Data

```
# Grab a peek at the data
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670

Code : Describing the Data

```
# Print the shape of the data
# data = data.sample(frac = 0.1, random_state = 48)
print(data.shape)
print(data.describe())
```

Output :

(284807, 31)

	Time	V1	...	Amount	Class
count	284807.000000	2.848070e+05	...	284807.000000	284807.000000
mean	94813.859575	3.919560e-15	...	88.349619	0.001727
std	47488.145955	1.958696e+00	...	250.120109	0.041527
min	0.000000	-5.640751e+01	...	0.000000	0.000000
25%	54201.500000	-9.203734e-01	...	5.600000	0.000000
50%	84692.000000	1.810880e-02	...	22.000000	0.000000
75%	139320.500000	1.315642e+00	...	77.165000	0.000000
max	172792.000000	2.454930e+00	...	25691.160000	1.000000

[8 rows x 31 columns]

Code : Imbalance in the data

Time to explain the data we are dealing with.

```
# Determine number of fraud cases in dataset
fraud = data[data['Class'] == 1]
valid = data[data['Class'] == 0]
outlierFraction = len(fraud)/float(len(valid))
print(outlierFraction)
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
```

```
0.0017304750013189597
Fraud Cases: 492
Valid Transactions: 284315
```

Only 0.17% fraudulent transaction out all the transactions. The data is highly Unbalanced. Lets first apply our models without balancing it and if we don't get a good accuracy then we can find a way to balance this dataset. But first, let's implement the model without it and will balance the data only if needed.

Code : Print the amount details for Fraudulent Transaction

```
print("Amount details of the fraudulent transaction")
fraud.Amount.describe()
```

Output :

Amount details of the fraudulent transaction

```
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

Code : Print the amount details for Normal Transaction

```
print("details of valid transaction")
valid.Amount.describe()
```

Output :

Amount details of valid transaction

```
count    284315.000000
mean       88.291022
std       250.105092
min        0.000000
25%        5.650000
50%       22.000000
75%       77.050000
max      25691.160000
```

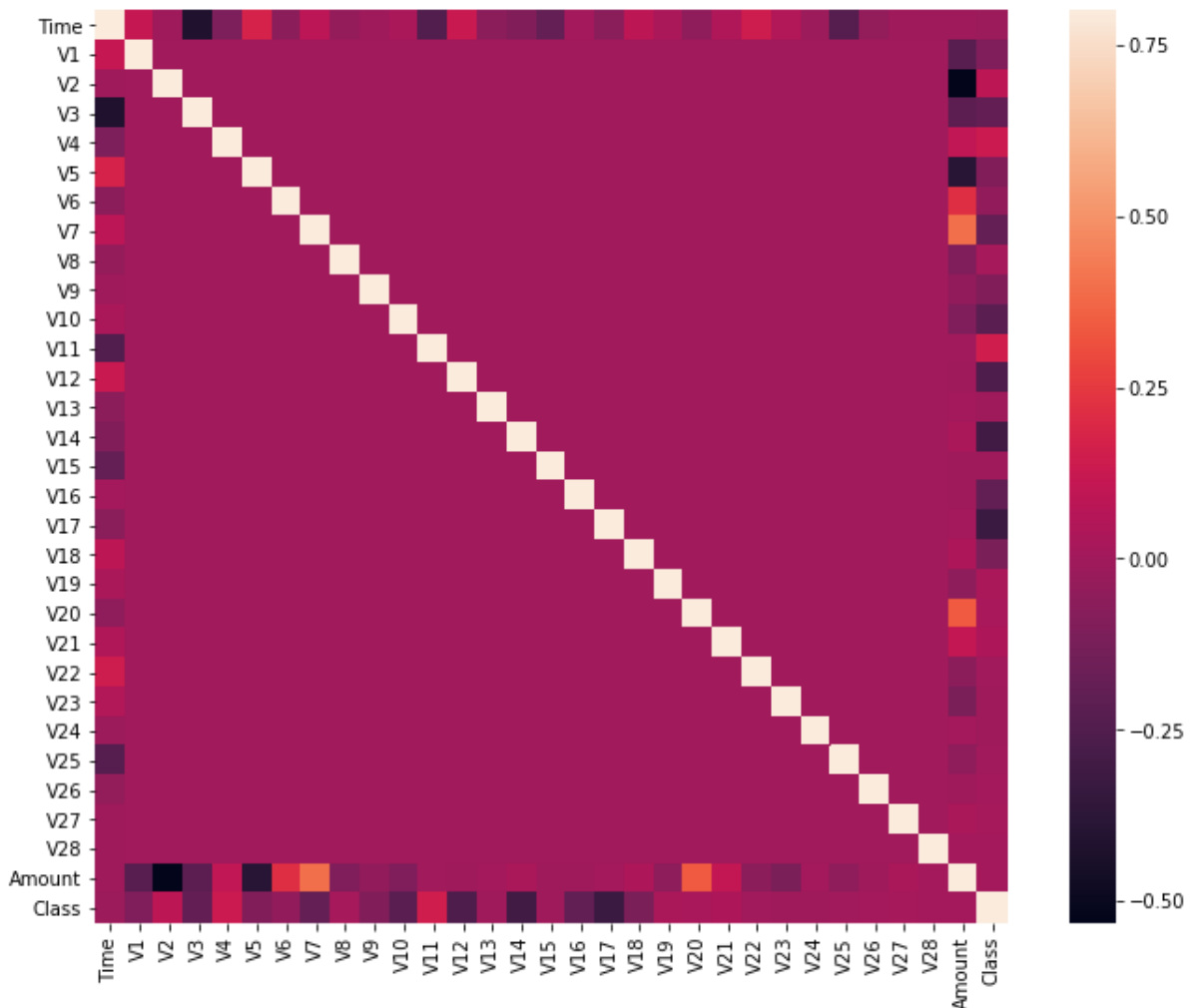
Name: Amount, dtype: float64

As we can clearly notice from this, the average Money transaction for the fraudulent ones is more. This makes this problem crucial to deal with.

Code : Plotting the Correlation Matrix

The correlation matrix graphically gives us an idea of how features correlate with each other and can help us predict what are the features that are most relevant for the prediction.

```
# Correlation matrix
cormat = data.corr()
fig = plt.figure(figsize = (12, 9))
sns.heatmap(cormat, vmax = .8, square = True)
plt.show()
```



In the HeatMap we can clearly see that most of the features do not correlate to other features but there are some features that either has a positive or a negative correlation with each other. For example, V2 and V5 are highly negatively correlated with the feature called *Amount*. We also see some correlation with V20 and *Amount*. This gives us a deeper understanding of the Data available to us.

Code : Separating the X and the Y values

Dividing the data into inputs parameters and outputs value format

```
# dividing the X and the Y from the dataset
X = data.drop(['Class'], axis = 1)
Y = data["Class"]
print(X.shape)
print(Y.shape)
# getting just the values for the sake of processing
# (its a numpy array with no columns)
xData = X.values
yData = Y.values
```

Output :

(284807, 30)

(284807,)

Training and Testing Data Bifurcation

We will be dividing the dataset into two main groups. One for training the model and the other for Testing our trained model's performance.

```
# Using Scikit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
xTrain, xTest, yTrain, yTest = train_test_split(
    xData, yData, test_size = 0.2, random_state = 42)
```

Code : Building a Random Forest Model using scikit learn

```
# Building the Random Forest Classifier (RANDOM FOREST)
from sklearn.ensemble import RandomForestClassifier
# random forest model creation
rfc = RandomForestClassifier()
rfc.fit(xTrain, yTrain)
# predictions
yPred = rfc.predict(xTest)
```

Code : Building all kinds of evaluating parameters

```
# Evaluating the classifier
# printing every score of the classifier
# scoring in anything
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, matthews_corrcoef
from sklearn.metrics import confusion_matrix

n_outliers = len(fraud)
n_errors = (yPred != yTest).sum()
print("The model used is Random Forest classifier")

acc = accuracy_score(yTest, yPred)
print("The accuracy is {}".format(acc))

prec = precision_score(yTest, yPred)
print("The precision is {}".format(prec))

rec = recall_score(yTest, yPred)
print("The recall is {}".format(rec))

f1 = f1_score(yTest, yPred)
print("The F1-Score is {}".format(f1))
```

```
MCC = matthews_corrcoef(yTest, yPred)
print("The Matthews correlation coefficient is{ }".format(MCC))
```

Output :

The model used is Random Forest classifier

The accuracy is 0.9995611109160493

The precision is 0.9866666666666667

The recall is 0.7551020408163265

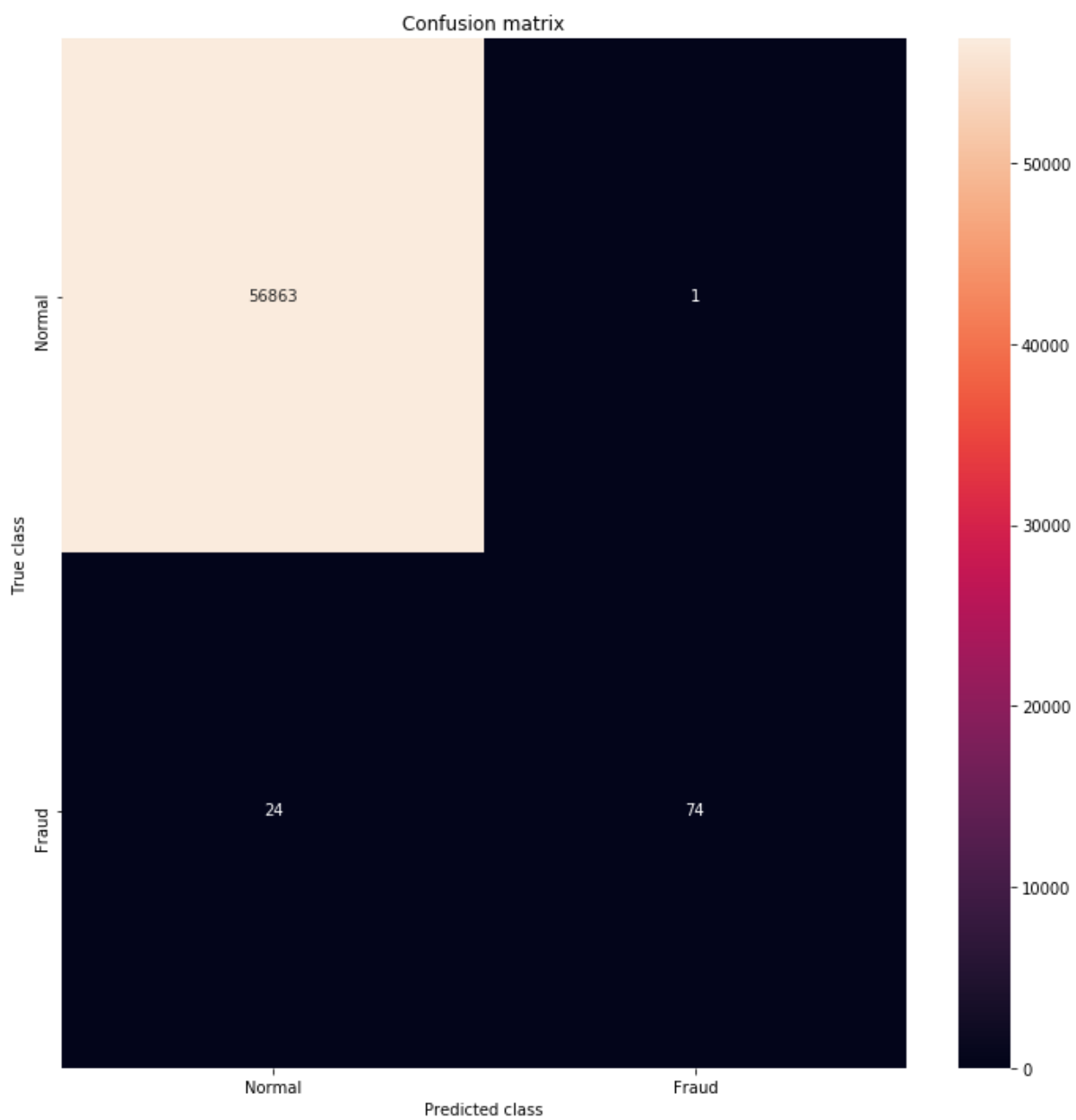
The F1-Score is 0.8554913294797689

The Matthews correlation coefficient is0.8629589216367891

Code : Visualizing the Confusion Matrix

```
# printing the confusion matrix
LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(yTest, yPred)
plt.figure(figsize =(12, 12))
sns.heatmap(conf_matrix, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix")
plt.ylabel("True class")
plt.xlabel("Predicted class")
plt.show()
```

Output :



Comparison with other algorithms without dealing with the imbalancing of the data.

What other Data Scientists got

Method Used	Frauds	Genuines	MCC
Naïve Bayes	83.130	97.730	0.219
Decision Tree	81.098	99.951	0.775
Random Forest	42.683	99.988	0.604
Gradient Boosted Tree	81.098	99.936	0.746
Decision Stump	66.870	99.963	0.711
Random Tree	32.520	99.982	0.497
Deep Learning	81.504	99.956	0.787
Neural Network	82.317	99.966	0.812
Multi Layer Perceptron	80.894	99.966	0.806
Linear Regression	54.065	99.985	0.683
Logistic Regression	79.065	99.962	0.786
Support Vector Machine	79.878	99.972	0.813

As you can see with our Random Forest Model we are getting a better result even for the recall which is the most tricky part.

- From the above process we define the steps through which we can detect a credit card fraud with the help of machine learning.
- As the above system we define are manual, In the given case every time we have to perform the fraud detection system manually and as we know that there are vast number of fraud occurring daily, Hence to solve this problem we have to automate the system so that it can perform its task automatically by learning from itself mistakes.
- But in the case of real life situations we have to let the machine perform the system of catching the fraud by automation.
- Hence below are the steps given for performing the automation of the credit card fraud system.

Automating the Machine Learning Pipeline for Credit card fraud detection

- Difficulty Level : [Medium](#)
- Last Updated : 03 May, 2020

Before going to the code it is requested to work on a Jupyter notebook or ipython notebook. If not installed on your machine you can use [Google Collab](#). This is one of the best and my personal favorite way of working on a python script to work on a Machine Learning problem

Dataset link:

You can download the dataset from this [link](#)

If the link is not working please go to this link and login to Kaggle to download the dataset.

Previous article: [Credit Card Fraud Detection using Python](#)

Now, I am considering that you have read the previous article without cheating, so let's proceed further. In this article, I will be using a library known as [Pycaret](#) that does all the heavy lifting for me and let me compare the best models side by side with just a few lines of code, which if you remember the first article took us a hell lot of code and all eternity to compare. We also able to do the most cumbersome job in this galaxy other than maintaining 75% attendance, hyperparameter tuning, that takes days and lots of code in just a couple of minutes with a couple of lines of code. It won't be wrong if you say that this article will be a short and most effective article you will read in a while. So sit back and relax and let the fun begin.

First install the one most important thing that you will need in this article, [Pycaret Library](#). This library is going to save you a ton of money as you know time is money, right.

To install the lib within your Ipython notebook use –

```
pip install pycaret
```

Code: Importing the necessary files

```
# importing all necessary libraries
```

```
# linear algebra
```

```
import numpy as np
# data processing, CSV file I / O (e.g. pd.read_csv)
import pandas as pd
```

Code: Loading the dataset

```
# Load the dataset from the csv file using pandas
# best way is to mount the drive on colab and
# copy the path for the csv file
path = "credit.csv"
data = pd.read_csv(path)
data.head()
```

Code: Knowing the dataset

```
# checking for the imbalance
len(df[df['Class']== 0])

len(df[df['Class']== 1])
```

Code: Setting up the pycaret classification

```
# Importing module and initializing setup
from pycaret.classification import * clf1 = setup(data = df, target = 'Class')
```

After this, a confirmation will be required to proceed. Press **Enter** for moving forward with the code.

Check if all the parameters type is correctly identified by the library.

Tell the classifier the percentage of training and validation split is to be taken. I took 80% training data which is quite common in machine learning.

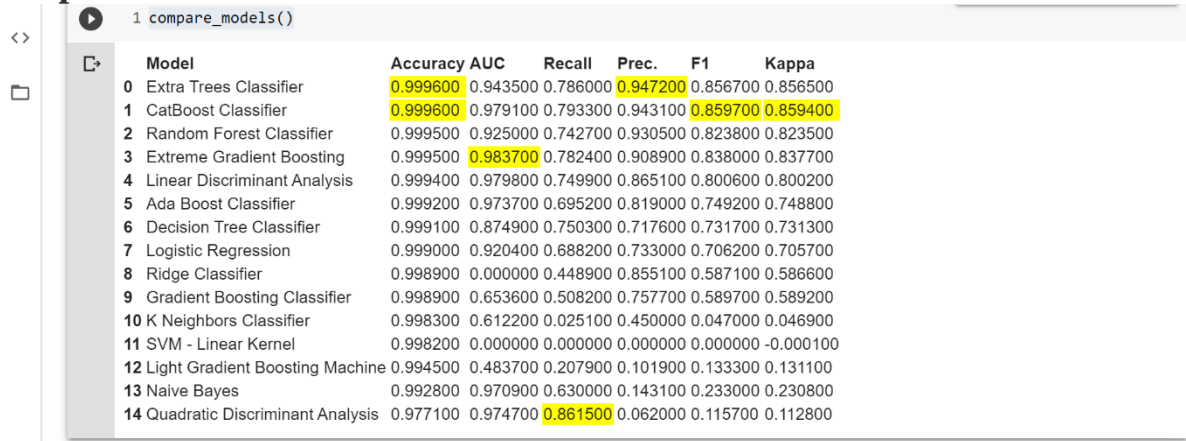
Coming to the next cell, this is the most important feature of the library. It allows the training data to be fit and compare to all the algorithms in the library to choose the best one. It displays which model is best and in what evaluation matrix. When the data is imbalance accuracy not always tell you the real story. I checked the precision but AUC, F1 and Kappa score can also be of great help to analyze the models. But this is going to an article amongst itself.

Code: Comparing the model

```
# command used for comparing all the models available in the library
```

```
compare_models()
```

Output:



	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	Extra Trees Classifier	0.999600	0.943500	0.786000	0.947200	0.856700	0.856500
1	CatBoost Classifier	0.999600	0.979100	0.793300	0.943100	0.859700	0.859400
2	Random Forest Classifier	0.999500	0.925000	0.742700	0.930500	0.823800	0.823500
3	Extreme Gradient Boosting	0.999500	0.983700	0.782400	0.908900	0.838000	0.837700
4	Linear Discriminant Analysis	0.999400	0.979800	0.749900	0.865100	0.800600	0.800200
5	Ada Boost Classifier	0.999200	0.973700	0.695200	0.819000	0.749200	0.748800
6	Decision Tree Classifier	0.999100	0.874900	0.750300	0.717600	0.731700	0.731300
7	Logistic Regression	0.999000	0.920400	0.688200	0.733000	0.706200	0.705700
8	Ridge Classifier	0.998900	0.000000	0.448900	0.855100	0.587100	0.586600
9	Gradient Boosting Classifier	0.998900	0.653600	0.508200	0.757700	0.589700	0.589200
10	K Neighbors Classifier	0.998300	0.612200	0.025100	0.450000	0.047000	0.046900
11	SVM - Linear Kernel	0.998200	0.000000	0.000000	0.000000	0.000000	-0.000100
12	Light Gradient Boosting Machine	0.994500	0.483700	0.207900	0.101900	0.133300	0.131100
13	Naive Bayes	0.992800	0.970900	0.630000	0.143100	0.233000	0.230800
14	Quadratic Discriminant Analysis	0.977100	0.974700	0.861500	0.062000	0.115700	0.112800

Yellow part is the top score for the corresponding model.

Taking a single algorithm performing decently in the comparison and creating a model for the same. The name of the algorithm can be found in the documentation of the [pycaret library under creating model](#)

Code: Creating the best model

```
# creating logistic regression model
```

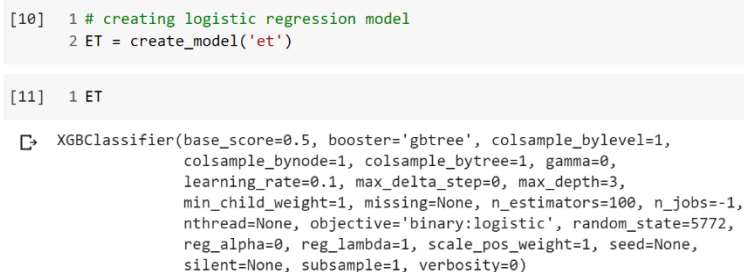
```
ET = create_model('et')
```

Code: Displaying the model parameters

```
# displaying the model parameters
```

```
ET
```

Output:



```
[10] 1 # creating logistic regression model
      2 ET = create_model('et')
```

```
[11] 1 ET
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=5772,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=0)
```

Code: Hyperparameter Tuning

```
# hyperparameter tuning for a particular model
```

```
model = tune_model('ET')
```

Output:

```
[12] 1 #hyperparameter tuning for a particular model
      2 model=tune_model('ET')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	0.9997	0.9823	0.8929	0.9259	0.9091	0.9089
1	0.9991	0.8921	0.5556	0.8824	0.6818	0.6814
2	0.9996	0.9941	0.8519	0.8846	0.8679	0.8677
3	0.9995	0.9628	0.7407	0.9524	0.8333	0.8331
4	0.9997	0.9957	0.9259	0.8929	0.9091	0.9089
5	0.9994	0.9973	0.6786	0.9500	0.7917	0.7914
6	0.9996	0.9800	0.7857	0.9565	0.8627	0.8625
7	0.9996	0.9933	0.7857	0.9565	0.8627	0.8625
8	0.9997	0.9912	0.8214	1.0000	0.9020	0.9018
9	0.9994	0.9743	0.7143	0.9524	0.8163	0.8160
Mean	0.9995	0.9763	0.7753	0.9354	0.8437	0.8434
SD	0.0002	0.0299	0.1036	0.0363	0.0656	0.0657

Code: Saving the model

After hours and hours of training the model and hyper tuning it, the worst thing that can happen to you is that the model disappears as the session time-out occurs. To save you from this nightmare, let me give a trick you will never forget.

```
# saving the model
```

```
save_model(ET, 'ET_saved')
```

Code: Loading the model

```
# Loading the saved model
```

```
ET_saved = load_model('ET_saved')
```

Output:

```
[17] 1 # saving the model
      2 save_model(ET, 'ET_saved')
```

Transformation Pipeline and Model Successfully Saved

```
[19] 1 # Loading the saved model
      2 ET_saved = load_model('ET_saved')
```

Transformation Pipeline and Model Successfully Loaded

Code: Finalizing the Model

A step just before deployment when you merge the train and the validation data and train model on all the data available to you.

```
# finalize a model
```

```
final_rf = finalize_model(rf)
```

Deploying the model is deployed on AWS. For the settings required for the same please visit the [documentation](#)

Deploy a model

```
deploy_model(final_lr, model_name = 'lr_aws', platform = 'aws', authentication = {  
    'bucket' : 'pycaret-test' })
```