Project Title: Credit Card Fraud Detection

Introduction:

In the last decade, there has been an exponential growth of the Internet. This has sparked the proliferation and increase in the use of services such as e-commerce, tap and pay systems, online bills payment systems etc. As a consequence, fraudsters have also increased activities to attack transactions that are made using credit cards. There exists a number of mechanisms used to protect credit cards transactions including credit card data encryption and tokenization. Although such methods are effective in most of the cases, they do not fully protect credit card transactions against fraud.

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.

Machine Learning (ML) is a sub-field of Artificial Intelligence (AI) that allows computers to learn from previous experience (data) and to improve on their predictive abilities without explicitly being programmed to do so. In this work we implement Machine Learning (ML) methods for credit card fraud detection. Credit card fraud is defined as a fraudulent transaction (payment) that is made using a credit or debit card by an unauthorized user. According to the Federal Trade Commission (FTC), there were about 1579 data breaches amounting to 179 million data points whereby credit card fraud activities were the most prevalent. Therefore, it is crucial to implement an effective credit card fraud detection method that is able to protect users from financial loss. One of the key issues with applying ML approaches to the credit card fraud detection problem is that most of the published work are impossible to reproduce. This is because credit card transactions are highly confidential. Therefore, the datasets that are used to develop ML models for credit card fraud detection contain anonymized attributes. Furthermore, credit card fraud detection is a challenging task because of the constantly changing nature and patterns of the fraudulent transactions. Additionally, existing ML models for credit card fraud detection suffer from a low detection accuracy and are not able to solve the highly skewed nature of credit card fraud datasets. Therefore, it is essential to develop ML models that can perform optimally and that can detect credit card fraud with a high accuracy score.

The recent advances of e-commerce and e-payment systems have sparked an increase in financial fraud cases such as credit card fraud. It is therefore crucial to implement mechanisms that can detect the credit card fraud. Features of credit card frauds play important role when machine learning is used for credit card fraud detection, and they must be chosen properly. This Project proposes a Random Forest classifier algorithm for feature selection. The Accuracy achieved is 0.9997 percent.

Project Code and Explanation

Our approach to building the classifier is discussed in the steps:

1. Perform Exploratory Data Analysis (EDA) on our dataset
2. Apply different Machine Learning algorithms to our dataset
3. Train and Evaluate our models on the dataset and pick the best one.

Step 1. Perform Exploratory Data Analysis (EDA)

There are a total of 284,807 transactions with only 492 of them being fraud. Let's import the necessary modules, load our dataset, and perform EDA on our dataset. Here is a peek at our dataset:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec


# 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("creditcard.csv")
```

Now, check for null values in the credit card dataset. Luckily, there aren't any null or NaN values in our dataset.

```
# Grab a peek at the data
data.head()
data.isnull().values.any()
# fill the NAN values by ffill method
```

```
data.fillna(method = 'ffill', inplace = True)
data.fillna(method = 'ffill', inplace = True)
# Print the shape of the data
# data = data.sample(frac = 0.1, random_state = 48)
print(data.shape)
print(data.describe())
```

Now, let’s check the number of occurrences of each class label and plot the information using matplotlib.

```
# 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])))
print('Amount details of the fraudulent transaction')
fraud.Amount.describe()


print('details of valid transaction')
valid.Amount.describe()

# To convert string to float value

data=data.astype(float)
#for i, row in data.iterrows():
 #   for column in data.columns:
  #      if type(row[column]) == str:
   #          print(f"String value found in row {i+1}, column '{column}': {row[column]}")
# Correlation matrix
corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))
```

```
sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()


# 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
```

Step 2: Apply Machine Learning Algorithms to Credit Card Dataset

Let’s train different models on our dataset and observe which algorithm works better for our problem. This is actually a binary classification problem as we have to predict only 1 of the 2 class labels. We can apply a variety of algorithms for this problem like Random Forest, Decision Tree, Support Vector Machine algorithms, etc.

In this machine learning project, we build Random Forest and Decision Tree classifiers and see which one works best. We address the “class imbalance” problem by picking the best-performed model.

But before we go into the code, let’s understand what random forests and decision trees are.

The Decision Tree algorithm is a supervised machine learning algorithm used for classification and regression tasks. The algorithm’s aim is to build a training model that predicts the value of a target class variable by learning simple if-then-else decision rules inferred from the training data.

Random forest (one of the most popular algorithms) is a supervised machine learning algorithm. It creates a “forest” out of an ensemble of “decision trees”, which are normally trained using the “bagging” technique. The bagging method’s basic principle is that combining different learning models improves the outcome.

```
# 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)

# 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)
```

Step 3: Train and Evaluate our Models on the Dataset

Now, Let’s train and evaluate the newly created models on the dataset and pick the best one.

Train the decision tree and random forest models on the dataset using the fit() function. Record the predictions made by the models using the predict() function and evaluate.

Let’s visualize the scores of each of our credit card fraud classifiers.

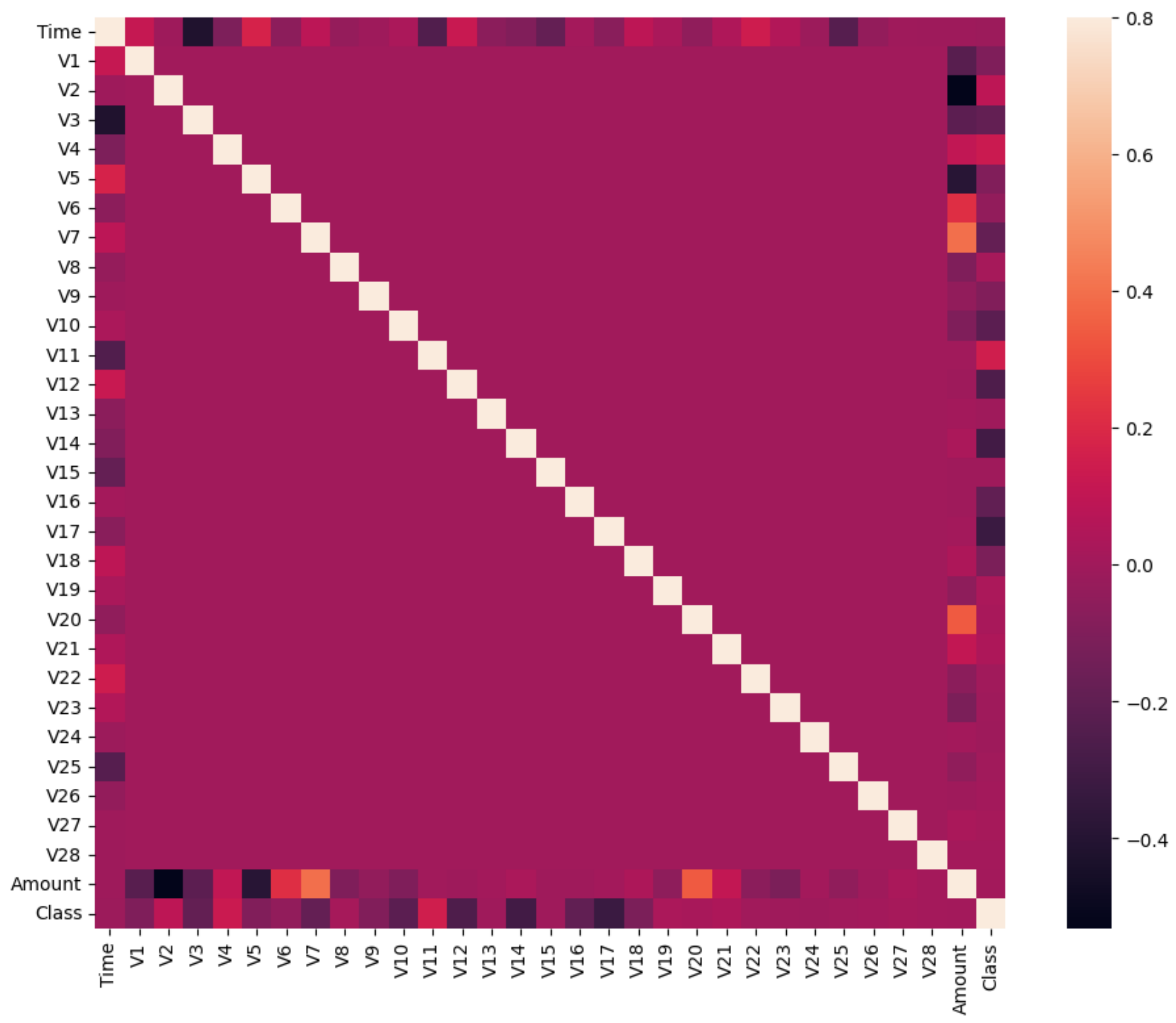
```
# Evaluating the classifier
# printing every score of the classifier
# scoring in anything
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score, matthews_corrcoef
from sklearn.metrics import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


# 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:-

```
                Time              V1              V2              V3            V4  \
count    284807.000000   2.848070e+05   2.848070e+05   2.848070e+05   2.848070e+05
mean      94813.859575   1.168375e-15   3.416908e-16  -1.379537e-15   2.074095e-15
std       47488.145955   1.958696e+00   1.651309e+00   1.516255e+00   1.415869e+00
min           0.000000  -5.640751e+01 -7.271573e+01  -4.832559e+01  -5.683171e+00
25%       54201.500000  -9.203734e-01 .- 5.985499e-01 -8.903648e-01  -8.486401e-01
50%       84692.000000   1.810880e-02   6.548556e-02   1.798463e-01  -1.984653e-02
75%      139320.500000   1.315642e+00   8.037239e-01   1.027196e+00   7.433413e-01
max      172792.000000   2.454930e+00   2.205773e+01   9.382558e+00   1.687534e+01

                  V5              V6              V7              V8            V9  \
count    2.848070e+05   2.848070e+05   2.848070e+05   2.848070e+05   2.848070e+05
mean     9.604066e-16   1.487313e-15  -5.556467e-16   1.213481e-16  -2.406331e-15
std      1.380247e+00   1.332271e+00   1.237094e+00   1.194353e+00   1.098632e+00
min     -1.137433e+02  -2.616051e+01  -4.355724e+01  -7.321672e+01  -1.343407e+01
25%     -6.915971e-01  -7.682956e-01  -5.540759e-01  -2.086297e-01  -6.430976e-01
50%     -5.433583e-02  -2.741871e-01   4.010308e-02   2.235804e-02  -5.142873e-02
75%      6.119264e-01   3.985649e-01   5.704361e-01   3.273459e-01   5.971390e-01
max      3.480167e+01   7.330163e+01   1.205895e+02   2.000721e+01   1.559499e+01

       ...           V21            V22             V23             V24  \
count  ...   2.848070e+05   2.848070e+05   2.848070e+05   2.848070e+05
mean   ...   1.654067e-16  -3.568593e-16   2.578648e-16   4.473266e-15
std    ...   7.345240e-01   7.257016e-01   6.244603e-01   6.056471e-01
min    ...  -3.483038e+01  -1.093314e+01  -4.480774e+01  -2.836627e+00
25%    ...  -2.283949e-01  -5.423504e-01  -1.618463e-01  -3.545861e-01
50%    ...  -2.945017e-02   6.781943e-03  -1.119293e-02   4.097606e-02
75%    ...   1.863772e-01   5.285536e-01   1.476421e-01   4.395266e-01
max    ...   2.720284e+01   1.050309e+01   2.252841e+01   4.584549e+00
```

```
                 V25            V26             V27             V28          Amount  \
count     2.848070e+05   2.848070e+05   2.848070e+05    2.848070e+05     284807.000000
mean      5.340915e-16   1.683437e-15   -3.660091e-16   -1.227390e-16    88.349619
std       5.212781e-01   4.822270e-01    4.036325e-01    3.300833e-01    250.120109
min      -1.029540e+01  -2.604551e+00   -2.256568e+01   -1.543008e+01     0.000000
25%      -3.171451e-01  -3.269839e-01   -7.083953e-02   -5.295979e-02     5.600000
50%       1.659350e-02  -5.213911e-02    1.342146e-03    1.124383e-02     22.000000
75%      3.507156e-01    2.409522e-01     9.104512e-02    7.827995e-02    77.165000
max       7.519589e+00   3.517346e+00     3.161220e+01    3.384781e+01     25691.160000

               Class
count       284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
0.0017304750013189597
Fraud Cases: 492
Valid Transactions: 284315
Amount details of the fraudulent transaction
```



details of valid transaction

(284807, 30)

(284807,)

The model used is Random Forest classifier

The accuracy is 0.9995786664794073

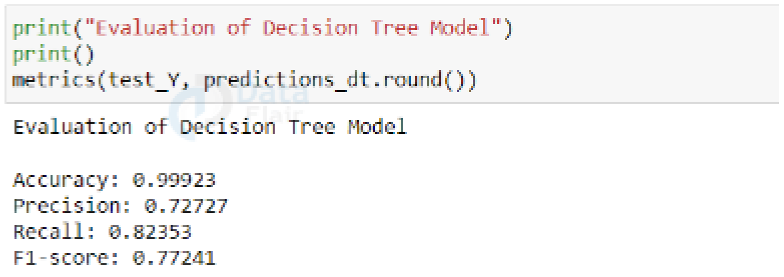
The precision is 0.9743589743589743

The recall is 0.7755102040816326

The F1-Score is 0.8636363636363635

The Matthews correlation coefficient is0.8690748763736589

Final Output by Decision Tree method


```
print("Evaluation of Decision Tree Model")
print()
metrics(test_Y, predictions_dt.round())
```

```
Evaluation of Decision Tree Model

Accuracy: 0.99923
Precision: 0.72727
Recall: 0.82353
F1-score: 0.77241
```

As the Random Forest algorithm performed better than the Decision Tree algorithm by 0.0003 percent.

Steps to Execute the Project

1. **Open https://colab.research.google.com/**
2. **upload the creditcard.csv file**
3. **copy the program in the Code Section**
4. **change Runtime type to GPU**
5. **click runtime and select run all, then output will be displayed**