Configuration Manual

MSc Research Project
MSc in Data Analytics

Purnima Duggal
Student ID: x20237928

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Purnima Duggal
Student ID:	20237928
Programme:	Data Analytics **Year:** 2022
Module:	MSc. Research Project
Lecturer:	Vladimir Milosavljevic
Submission Due Date:	15-08-2022
Project Title:	Predicting Credit Card Fraud Using Conditional Generative Adversarial Network
Word Count	1385 **Page Count** 14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: .....

Date: .....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Purnima Duggal
x20237928

1 Introduction

This paper aims to present a comprehensive outline of all the steps involved in conducting the research from setting up the environment to getting the results. We aim to improve the prediction of credit card fraud by using CT-GAN to address the issue of class identification. With the goal of addressing the class imbalance, we have used two approaches: SMOTE and CT-GAN. After applying SMOTE and CT-GAN we performed two experiments using three different classifiers namely: isolation forest, multilayer perceptron, and random forest.

2 System Specification

2.1 Hardware Specification

Table 1: Hardware Specifications

Operating System	Windows 10 Home
Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
RAM	8GB
System Type	64- bit Operating System, x64-based processor

2.2 Software Specification

We have used Google Collaboratory to conduct this research. Google Collab is a free coding tool by Google and gives free access to the cloud platform. The code is written in Python programming language having Version: Python 3.7.11

3 Environment Setup

The actions mentioned below must be followed in order to set up Google Collab:

1. Log in to your Gmail account using your credentials.

2. Now, go to Google Drive from the Google Apps menu and upload the dataset downloaded from Kaggle. This will take some time to upload.

3. Once, the data is uploaded to google drive, now we will go to google Collaboratory, where we will create a new notebook and also enable the GPU to run the code.I have created a folder on Google Drive with name ‘thesis_dt’.

4. Now using the below code, we will import our dataset from google drive by mounting the drive with google collab.

```
# IMPORTING DATA
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/thesis_dt/creditcard.csv")
```

```
Mounted at /content/drive
```

Figure 1: Importing dataset from Google Drive

5. After this, a dialog box will appear where Google will request for access to Google Drive, click on Allow.

4 Data Understanding

4.1 Dataset Setup

1. Download the Credit Card Fraud dataset from the Kaggle repository as seen in Figure.

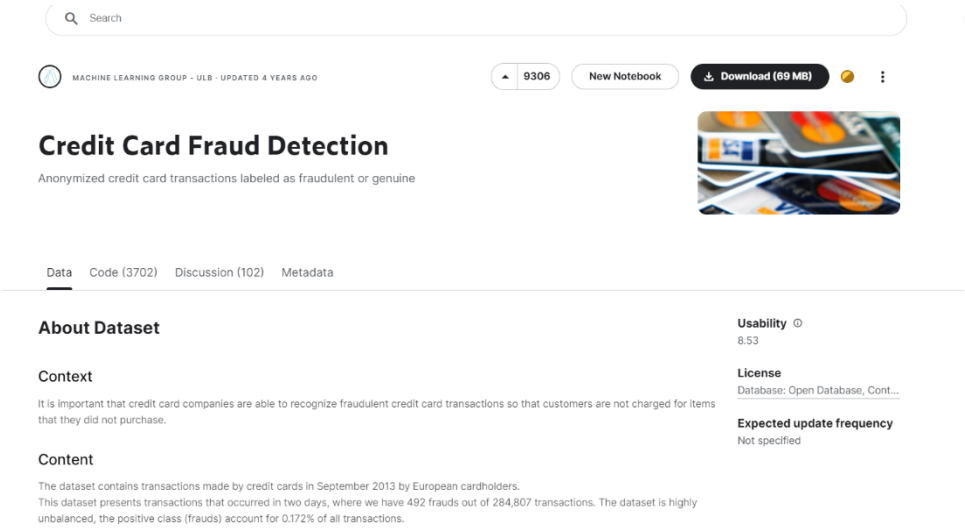

Figure 2: Credit Card Fraud Dataset

2. Now we follow the steps mentioned in Section 3 for setting up the coding environment.
3. We need to import the required libraries to be used in later sections.

```
# import packages
import numpy as np
import pandas
import seaborn
import matplotlib.pyplot as plt

# For GANs
from tensorflow import keras

# for PCA
from sklearn.decomposition import PCA

# For oversampling
from imblearn.over_sampling import SMOTE

# for scaling
from sklearn.preprocessing import MaxAbsScaler

# MLP model package
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Isolation Forest and Random Forest packages
from sklearn.ensemble import IsolationForest, RandomForestClassifier
```

Figure 3:Importing the required libraries

4.2 Data Exploration

1. We began by looking at the dataset's sample size, the number of columns, and data type of each column.

```
print('Sample size of dataset : ' + str(df.shape[0]))
print('Number of Columns present : ' + str(df.shape[1]))
for i in range(df.shape[1]) :
    print(df.columns[i], end = ' ')
```

```
Sample size of dataset : 284807
Number of Columns present : 31
Time V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 Amount Class
```

Figure 4: Dataset sample size and columns

```
Sample size : 284807
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 69.5 MB
```

Figure 5: Data Type of each variable

2. Before proceeding, we looked for missing values in both rows and columns and duplicate data.

Figure 6: Missing values

3. We found some duplicate values; these will be handled in data preparation stage.

```
# Check for duplicates

duplicate = df[df.duplicated()]
duplicate
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
33	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.711206	0.176066	-0.286717	...	0.046949	0.208105	-0.185548	0.001031	0.098816	-0.552904	-0.073288	0.023307	6.14	0
35	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.693039	0.179742	-0.285642	...	0.049526	0.206537	-0.187108	0.000753	0.098117	-0.553471	-0.078306	0.025427	1.77	0
113	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.036715	0.350995	0.118950	...	0.102520	0.605089	0.023092	-0.626463	0.479120	-0.166937	0.081247	0.001192	1.18	0
114	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.036715	0.350995	0.118950	...	0.102520	0.605089	0.023092	-0.626463	0.479120	-0.166937	0.081247	0.001192	1.18	0
115	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.036715	0.350995	0.118950	...	0.102520	0.605089	0.023092	-0.626463	0.479120	-0.166937	0.081247	0.001192	1.18	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
282987	171288.0	1.912550	-0.455240	-1.750654	0.454324	2.089130	4.160019	-0.881302	1.081750	1.022928	...	-0.524067	-1.337510	0.473943	0.616683	-0.283548	-1.084843	0.073133	-0.036020	11.99	0
283483	171627.0	-1.464380	1.368119	0.815992	-0.601282	-0.689115	-0.487154	-0.303778	0.884953	0.054065	...	0.287217	0.947825	-0.218773	0.082926	0.044127	0.639270	0.213565	0.119251	6.82	0
283485	171627.0	-1.457978	1.378203	0.811515	-0.603760	-0.711883	-0.471672	-0.282535	0.880654	0.052808	...	0.284205	0.949659	-0.216949	0.083250	0.044944	0.639933	0.219432	0.116772	11.93	0
284191	172233.0	-2.667936	3.160505	-3.355984	1.007845	-0.377397	-0.109730	-0.667233	2.309700	-1.639306	...	0.391483	0.266536	-0.079853	-0.096395	0.086719	-0.451128	-1.183743	-0.222200	55.66	0
284193	172233.0	-2.691642	3.123168	-3.339407	1.017018	-0.293095	-0.167054	-0.745886	2.325616	-1.634651	...	0.402639	0.259746	-0.086606	-0.097597	0.083693	-0.453584	-1.205466	-0.213020	36.74	0

Figure 7: Check for Duplicate Values

4. To understand the distribution of fraud and non-fraud transaction, we plot a pie chart.

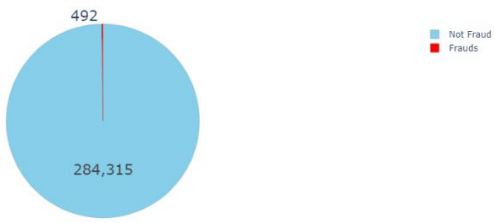

Figure 8: Pie chart representing Class Distribution

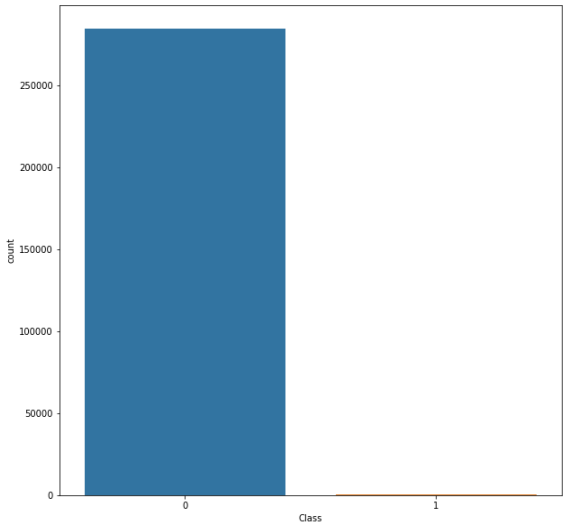

Figure 9: Count plot of dataset

5. We plot the graphs for the columns Time and Amount to observe the distribution of dataset.

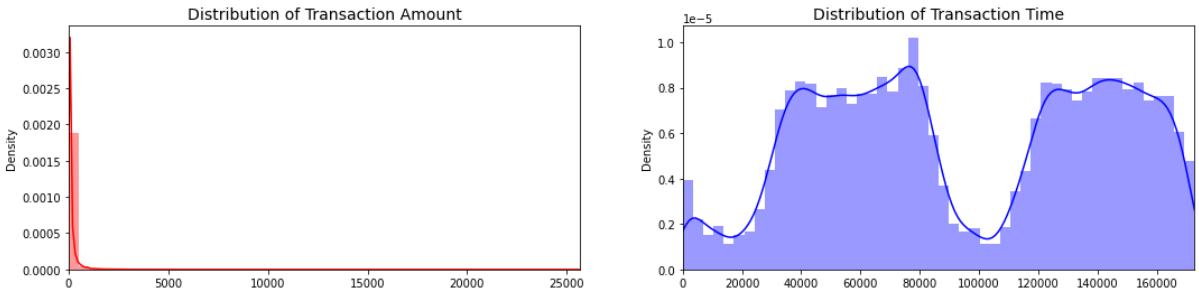

Figure 10: Distribution of Transaction Amount and Time

4.3 Data Preparation

1. We will start by handling the duplicate values present in Section 4.2

```
#Removing duplicates from the dataset
#Making a copy of dataset in card_d
card_d=df.copy()
card_d.drop_duplicates(subset=None, inplace=True)
```

```
#Checking the size of card_d after removing duplicate values
card_d.shape
```

```
(283726, 31)
```

```
#Checking size of original dataset
df.shape
```

```
(284807, 31)
```

```
#Assigning value of card_d to original dataset
df=card_d
```

Figure 11: Handling Duplicate values

2. As the data is highly skewed so we have used two approaches to handle this:

I. SMOTE Technique

1. We have uses SMOTE technique where it chooses two samples, where one sample is chosen from the minority class and one from KNN. We are using K_neighbours=5.

2. We have divided the data into tests and train to perform the SMOTE technique. We applied SMOTE technique to the training dataset X_over and Y-Over as seen in figure 12.

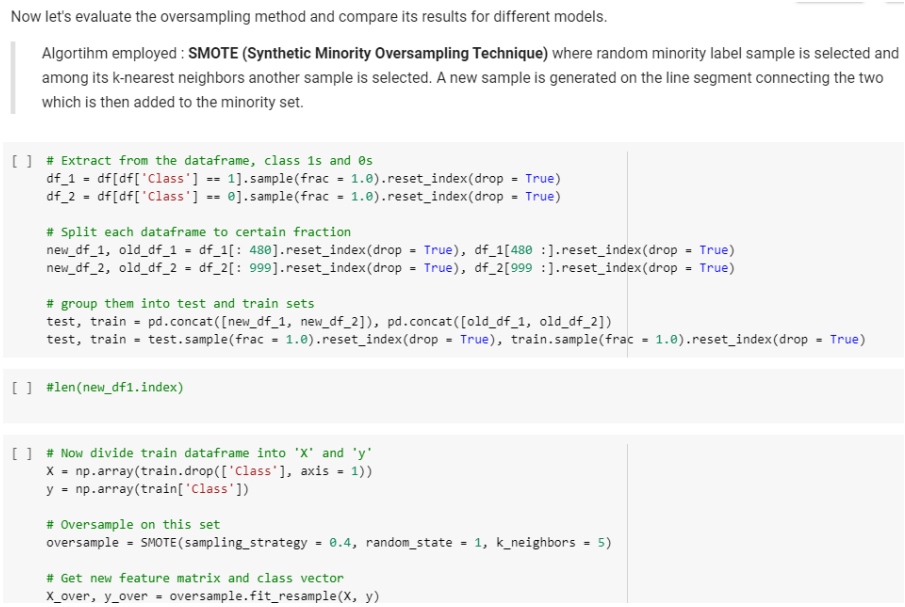

Figure 12: SMOTE Technique

II. CT-GAN Approach

1. We have used CT-GAN Approach to handle the class imbalance. We begin the code by defining the Discriminator and Generator.

```python
# Let's define the discriminator which takes inputs the feature
# matrix and class vector and predicts the probability of being
# fake or real (0 or 1)
def dis () :

    # define the feature input
    feature = keras.Input(shape = (30,))

    # define the labels  input
    labels  = keras.Input(shape = (1, ))

    # merge the two layers
    merge = keras.layers.Concatenate()([feature,labels])

    # add one hidden layer
    model = keras.layers.Dense(200)(merge)
    model = keras.layers.LeakyReLU(alpha = 0.2)(model)

    # add the output layer
    model = keras.layers.Dense(1,activation='sigmoid')(model)

    # create a model from the pipeline
    d_model = keras.Model(inputs = [feature,labels], outputs = model, name = 'discriminator')

    # compile the model
    d_model.compile(optimizer = keras.optimizers.Adam(learning_rate = 0.0002), loss = 'binary_crossentropy'
                    metrics = ['accuracy'])

    # return the model
    return d_model
```

```python
# Let's define the generator which takes as input latent
# space and class labels and ouputs a feature matrix 'X'
def gen () :

    # define the latent space
    latent = keras.Input(shape = (99,))
```

Figure 13 : Defining Discriminator and Generator

2. After defining, the discriminator and generator, we have combined them into CGAN

```python
# Combine the discriminator and Generator into cGAN
# Keep discriminator as non-trainable, so it does
# not update its weights during training the cgan.
def gan (discriminator, generator) :

    # Make discriminator as non-trainable
    discriminator.trainable = False

    # get inputs from the generator
    gen_latent, gen_labels  = generator.input

    # get output from the generator
    gen_feature = generator.output

    # get discriminator predictions
    dis_predict = discriminator([gen_feature, gen_labels])

    # create a model from the pipeline
    cgan_model  = keras.Model(inputs = [gen_latent, gen_labels], outputs = dis_predict, name = 'cGAN')

    # compile the model
    cgan_model.compile(optimizer = keras.optimizers.Adam(learning_rate = 0.0002), loss = 'binary_crossentropy')

    # return the model
    return cgan_model
```

Figure 14: Combining Generator and Discriminator to CGAN

3. Now, we have defined a function to generate fake samples of the data.

```
# Let's define a generate_fake_samples
# method to generate random samples as
# fake data from latent sapce
def generate_latent_point (n_samples) :

    # create an array of n_samples * late
    # -nt_space (= 99)
    Z_lat = np.random.randn(n_samples*99)
    X_lat = Z_lat.reshape(n_samples , 99)

    # randomly generate the class labels 'y'
    y_lat = np.random.randint(0, 2, n_samples)

    # return the points
    return [X_lat, y_lat]

def generate_fake_samples (generator, n_samples) :

    # generate n_samples points in latent space
    [X_lat, y_lat] = generate_latent_point(n_samples)

    # predict on them using the generator
    X_fake = generator.predict([X_lat, y_lat])

    # classify them as '0' (fake)
    y_labels = np.zeros((n_samples, 1))

    # return the computed arrays
    return [X_fake, y_lat], y_labels
```

Figure 15 : Generating Fake samples

4. We trained our cgan model till epochs=10.

```
[ ] # train the model
    def train (discriminator, generator, cgan, batch_size = 128, epochs = 10) :
        count = 0

        half_batch = int(batch_size/2)
        batch_per_epoch = int(X.shape[0]/batch_size)

        d_real = []
        d_fake = []
        g = []
        for i in range(epochs) :
            for j in range(batch_per_epoch) :

                # real
                [X_real, y_real], y_labels_real = generate_real_samples(half_batch)
                # loss on real
                d1_loss, _ = discriminator.train_on_batch([X_real, y_real], y_labels_real)
                # fake
                [X_fake, y_fake], y_labels_fake = generate_fake_samples(generator,
                                                                        half_batch)
                # loss on fake
                d2_loss, _ = discriminator.train_on_batch([X_fake, y_fake], y_labels_fake)
```

Figure 16: Training the model

5. We defined a function to plot the graph showing the fake samples, real data and cgan loss function with the generated data.

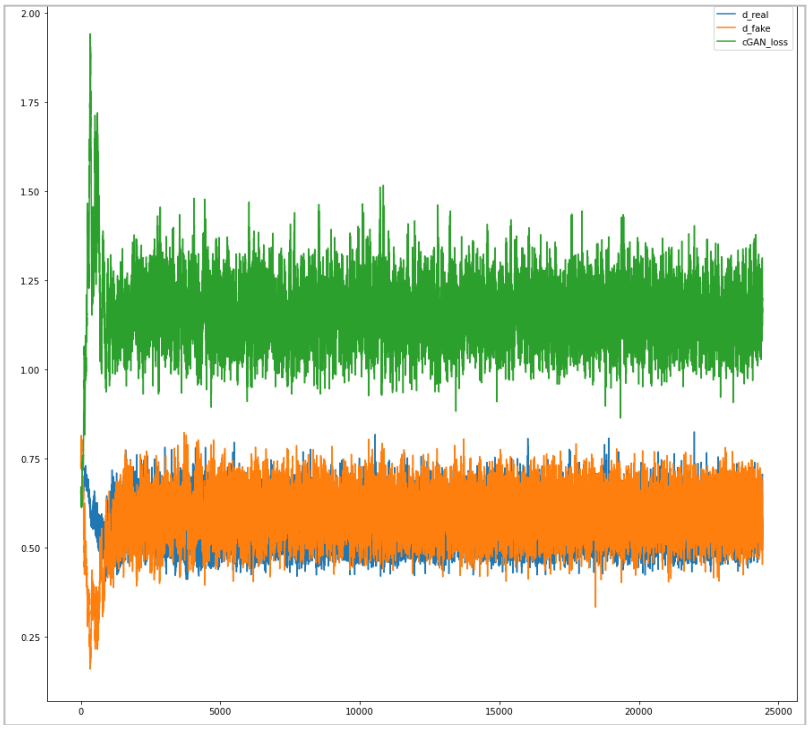

Figure 17: Data produced by CGAN

5 Modelling

We used three different models in this research. We first defined a dictionary to hold all the models so that it would be easy to just call the model by passing their function names.

I. Isolation Forest

```
# Isolation Forest and Random Forest packages
from sklearn.ensemble import IsolationForest, RandomForestClassifier
```

```
# Let's define the IsolationForest Model
def IsF (X, y, ratio) :

    # train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)

    # define the model
    model = IsolationForest(n_estimators = 20, contamination = ratio, random_state = 1)

    # train the model
    model.fit(X_train, y_train)

    # predict on the test set
    predictions = model.predict(X_test)

    # Convert the predictions according to problem profile
    predictions[predictions ==  1] = 0
    predictions[predictions == -1] = 1

    # evaluate the accuracy and classification report
    print('IsF : ' + str(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict['IsF'] = model
```

Figure 18: Isolation Forest

We start this model by importing the required libraries. We have used IsF() function for Isolation Forest to define the model, train the model and predict the model. We have divided the dataset into train and test as seen in Figure 18. We have used 20 estimators in this model and the value of the ratio is defined when this model is called. We have also declared accuracy and classification report function inside this method of the model.

II. Multi-Layer Perceptron

```
# Let's define the MultiLayer Perceptron
def MLP (X, y, param) :

    # train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)

    # hyper-parameters
    nodes = param['nodes']
    lrate = param['lrate']
    toler = param['toler']
    batch = param['batch_size']

    # define the model
    model = MLPClassifier(hidden_layer_sizes = (nodes,), tol = toler, batch_size = batch, learning_rate_init = lrate,
                          verbose = 0, random_state = 1)

    # train the model
    model.fit(X_train, y_train)

    # predict on the test set
    predictions = model.predict(X_test)

    # evaluate the accuracy and classification report
    print('MLP : ' + str(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict['MLP'] = model
```

Figure 19: Multi-Layer Perceptron

Multi-Layer Perceptron is also defined in the similar manner. We have used MLP() function for Multi-Layer Perceptron. We have defined the function by passing X, Y and the param object, the value of this object will be defined when this function MLP() will be called.

III. Random Forest

```
# Let's define the Random Forest Classifier Model
def RFR (X, y, split) :

    # train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)

    # define the model
    model = RandomForestClassifier(n_estimators = 100, criterion = 'gini', min_samples_split = split, random_state = 1)

    # train the model
    model.fit(X_train, y_train)

    # predict on the test set
    predictions = model.predict(X_test)

    # evaluate the accuracy and classification report
    print('RFR : ' + str(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))

    # assign this to models_dict
    models_dict['RFR'] = model
```

Figure 20: Random Forest

We defined Random Forest Classifier inside a function called RFR(), where we defined the model, train the model ,predict the model and evaluate the model. The split value is defined when the model is called before any class imbalance technique.

6 Evaluation

6.1 Experiment 1: Applying SMOTE

```
# Now divide train dataframe into 'X' and 'y'
X = np.array(train.drop(['Class'], axis = 1))
y = np.array(train['Class'])

# Oversample on this set
oversample = SMOTE(sampling_strategy = 0.4, random_state = 1, k_neighbors = 5)

# Get new feature matrix and class vector
X_over, y_over = oversample.fit_resample(X, y)
```

```
[ ] # Assigning important parameters for the models

    # For Isolation Forest Model
    ratio = float(y_over[y_over == 1].shape[0]/y_over.shape[0])

    # For Multi Layer Perceptron
    param = {
        'nodes' : 170,
        'lrate' : 0.00005,
        'toler' : 0.00001,
        'batch_size' : 100
    }

    # For Random Forest Classifier
    split = 2

    #For ADA
    obj=1
```

Figure 21: Applying SMOTE

1. We applied the SMOTE technique and passed the parameters into the models to train them first on the data and then test them.
2. To test the performance of all the three classifiers, we plot the AUPRC curve.

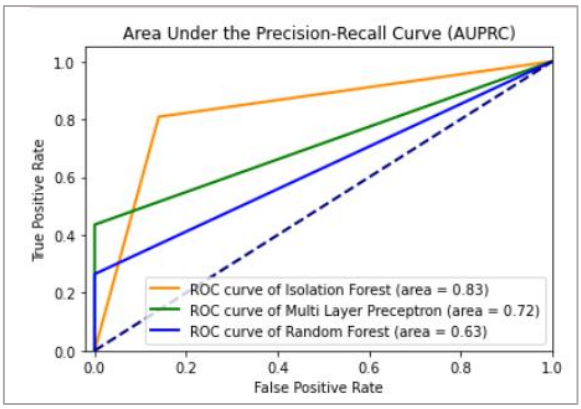

Figure 22: AUPRC graph after SMOTE

Isolation Forest performs the best as seen in figure 22.

6.2 Applying CT-GAN

```
[ ]  # parameters

     # For Isolation Forest Model
     ratio = float(y_gan[y_gan == 1].shape[0]/y_gan.shape[0])

     # For Multi Layer Perceptron
     param = {
         'nodes' : 5200,
         'lrate' : 1,
         'toler' : 0.01,
         'batch_size' : 200
     }

     # For Random Forest Classifier
     split = 2

     #For ADA
     obj=1
```

Figure 22: Applying CT-GAN

We applied the CT-GAN model to the three classifiers, by passing the Y_GAN variable to the model functions. On analyzing the AUPRC graph, again Isolation Forest outperforms other two models.

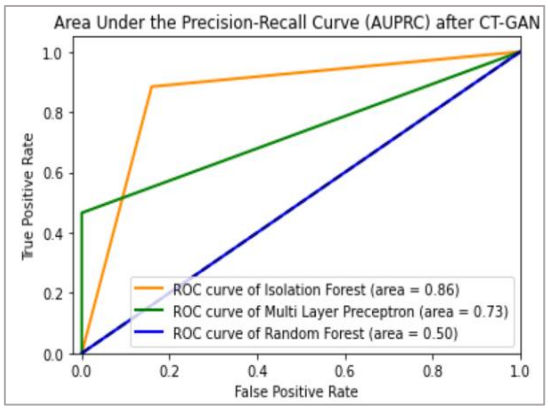

Figure 23: AUPRC graph after CT-GAN

7 Conclusion

In a thorough description, every step of the project's implementation is described. To make the code more readable, all Python code is written with the suitable comments.

References

Forough J.,Momtazi(2021).Sequential credit card fraud detection: A deep neural network and probabilistic graphical model approach.

Benchaji, I., Douzi, S., El Ouahidi, B. et al (2021). Enhanced credit card fraud detection based on attention mechanism and LSTM deep model. *J Big Data 8, 15.*