Project on Detecting Fraudulent Transactions using Machine Learning Models

by
DINESH KUMAR BOLISETTY
Enrl No: **40AIML154-21/1**
dineshkb@gmail.com

Problem Overview:

The current trend in the world is that more online payment transactions are being increased due to various reasons. As the online payment transactions are making the things easier and also making the various countries adopt digital money which is benefiting the respective countries to eliminate the actual money transactions which are not accounted for like tax and others. While these cashless payment transactions benefit various governments and people, at the same time we are suffering from cyber fraud. This project is about how we can improve the efficacy of finding fraudulent transactions and alerting the same with the help of the respective transaction data. This will help hundreds of thousands of businesses to reduce their loss and help to increase their revenue. Also it will help to win the customer/people's trust and encourage them to do more online cashless payments.

Data Set:

In this project we are planning to build a model which can predict the given transaction as fraudulent or not and alert the same in case if it is found to be fraud. To build the model we are planning to use the dataset provided in Kaggel as part of the competition held on "**IEEE-CIS Fraud Detection**". In this competition the dataset is provided by world's leading payment service company, Vesta Corporation, seeking the best solutions for fraud prevention.

Link for the dataset: https://www.kaggle.com/c/ieee-fraud-detection

The Kaggle data is broken into two files identity and transaction, which are joined by TransactionID. Not all transactions have corresponding identity information.

Features - Transaction Data
- TransactionID
- isFraud
- TransactionDT
- TransactionAmt

- ProductCD
- card1
- card2
- card3
- card4
- card5
- card6
- addr1
- addr2
- dist1
- dist2
- P_emaildomain
- R_emaildomain
- C1 - V339 (Anonymous Features)

Features - Identity Data

- TransactionID
- id_1 - id_38 (Anonymous Features)
- DeviceType
- DeviceInfo

The TransactionDT feature is a timedelta from a given reference datetime (not an actual timestamp). Detailed features and the data analysis will be done as part of EDA in the next phase.

The given data in Kaggle competition can be handled by using Pandas tool since it is moderate in size. The data has many features which are anonymous. Also the data is highly imbalanced based on the below stats. As the data was highly imbalanced there is a need to upsample or generate the synthetic data using SMOTE method based on the available data in primary source.

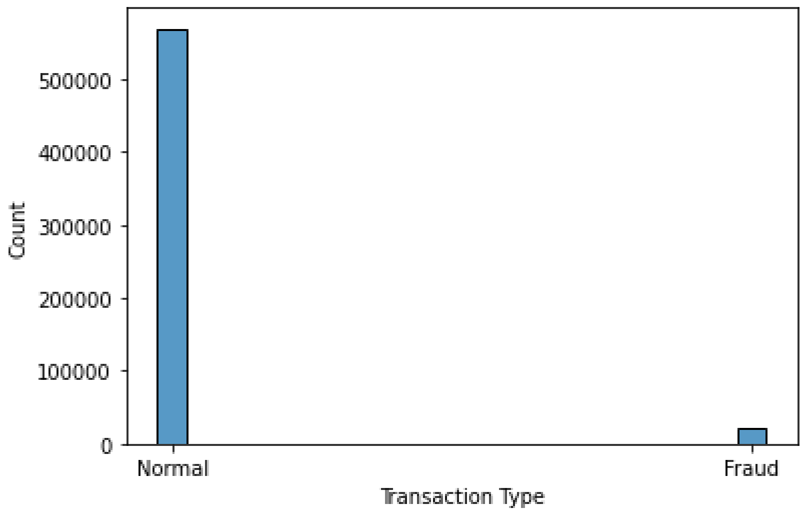


Fig. 1: Count of Fraudulent and Normal Transactions from the Kaggle Data Set

Key Metric(KPI) to Optimize:

The most commonly used evaluation metric in regression problems for classification is Log-Loss. The log-loss-error is the negative log of the probability of a datapoint belonging to the class as denoted by its label. Log loss is the average log error computed over a set of data points.

For binary classification, it is defined by the equation shown below:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Metric used to calculate the efficiency of the Model:

A useful tool when predicting the probability of a binary outcome is the Receiver Operating Characteristic curve, or ROC curve. The key business metric used to optimize will be the area under the RoC curve for calculating the efficacy of the model build to identify the fraudulent transactions. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings between 0.0 to 1.0.

The true positive rate is calculated as the number of true positives divided by the sum of the number of true positives and the number of false negatives. It describes how good the model is at predicting the positive class when the actual outcome is positive. The true-positive rate is also known as sensitivity, recall or probability of detection. The false-positive rate is also known as probability of false alarm and can be calculated as (1 − specificity). It can also be thought of as a plot of the power as a function of the Type I Error of the decision rule (when the performance is calculated from just a sample of the population, it can be thought of as estimators of these quantities). The ROC curve is thus the sensitivity or recall as a function of fall-out. The curves of different models can be compared directly in general or for different thresholds.

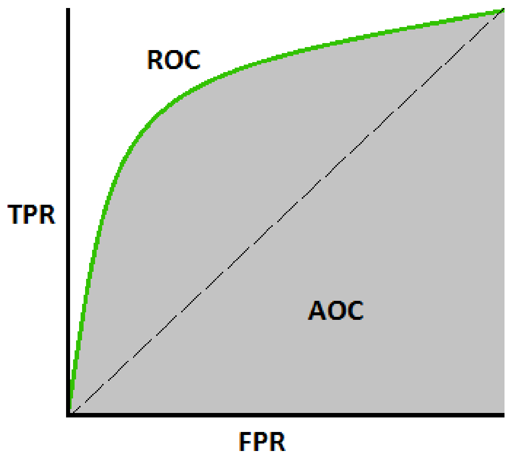


Fig. 2: ROC Curve Example (TPR vs FPR)

The area under the curve (AUC) can be used as a summary of the model skill. The shape of the curve contains a lot of information, including what we might care about most for a problem, the expected false positive rate, and the false negative rate. ROC curves do not provide a single-value performance score which motivates the use of AUC. The AUC allows the evaluation of the best model on average. Still, it is not biased towards the minority class.

The other alternate metrics that can be used is Confusion Matrix is a matrix that helps us visualize a model's prediction performance from multiple perspectives. It is a square matrix, whose columns represent predicted values and the rows represent actual values. The diagonal of the confusion matrix represents correct predictions. The following core performance metrics can be derived from the confusion matrix:

1. Accuracy
2. Precision of prediction with respect to each class
3. Recall of prediction with respect to each class
4. F1 Score
5. Precision - Recall Curve

Accuracy basically expresses the percentage of total correct predictions. The Precision of prediction with respect to a class is the percentage of correct predictions among the total predictions made for that class.The recall of prediction with respect to a class is the percentage of that class recovered within the total predictions made for that class.

ROC Metric will not work well for the below cases:

- ROC-AUC does not work well under severe imbalance in the dataset.
- ROC-AUC tries to measure if the rank ordering of classifications is correct it does not take into account actually predicted probabilities.

This metric is also used in case of health care for determining critical illness like cancer etc.

Real world challenges and constraints:

The challenges while solving the fraud detection problems are outlined as mentioned below:

1. Changing fraud patterns over time:
   The fraudsters are always in the lookout to find new and innovative ways to get around the systems to improve their actions. Hence it becomes all-important for the machine learning models to be updated with the evolved patterns to identify the fraud transactions. These new/updated ways result in a decrease in the model's performance and efficiency.
2. Class Imbalance:
   In practice only a small percentage of customers have fraudulent intentions. Hence, there is an imbalance in the classification of fraud detection models which makes it harder to build them.
3. Model Interpretations:
   This limitation is associated with the concept of explainability since models typically give a score indicating whether a transaction is likely to be fraudulent or not — without explaining why.
4. Feature generation can be time-consuming:
   Subject matter experts can require long periods of time to generate a comprehensive feature set which slows down the fraud detection process. Also, the given data or publicly available data like kaggle will have many anonymous features and makes it difficult to interpret them to generate a meaningful feature set.

Similar problems solved in literature:

These kinds of problems are resolved in literature by various methods. This problem is categorised in ML world as binary classification of transactions with feature isFraud "yes" / "no" or 1 / 0.

The below mentioned solutions approaches can be used to resolve this problem:

1. Logistic Regression - Binary Classification
2. Logistic Regression with different regularizations
3. Random Forest
4. GBDT
5. XG Boost DT
6. Light GBDT
7. Support Vector Machines (SVM)
8. Deep Learning for classification

EDA and Feature Extraction:

The data used in this project is downloaded from the Kaggle competition. As explained earlier the data is broken into two files identity and transaction, which are joined by TransactionID. Not all transactions have corresponding identity information.

Link for the dataset: https://www.kaggle.com/c/ieee-fraud-detection

The training data size is as mentioned below:
Transaction data shape: (590540, 394)
Identity data shape: (144233, 41)

Categorical Features - Transaction

- ProductCD
- card1 - card6
- addr1, addr2
- P_emaildomain
- R_emaildomain
- M1 - M9

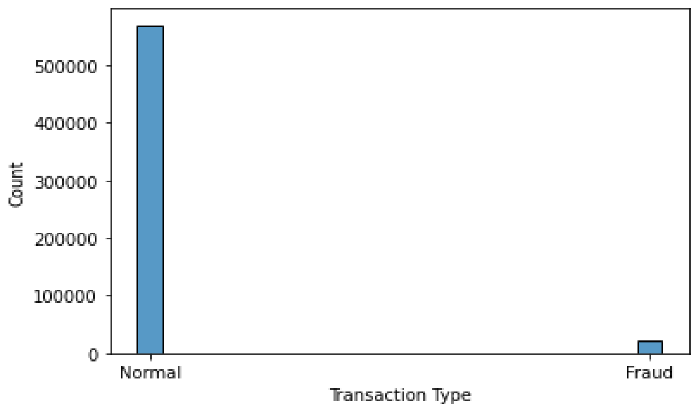
Categorical Features - Identity

- DeviceType
- DeviceInfo
- id_12 - id_38

The TransactionDT feature is a timedelta from a given reference datetime (not an actual timestamp).

Exploratory Data Analysis & Observations:

The transaction data was highly imbalanced in terms of normal and fraud transactions.



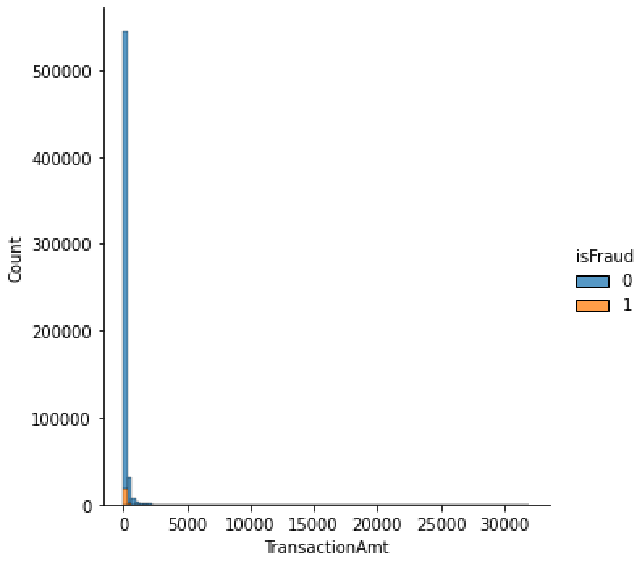
Count of Fraudulent and Normal Transactions from the Kaggle Data Set

The transaction data has a lot of missing data which are marked as NaN (null) values. A simple python code provided in iPython notebook to list out the number of null values from each column and **41.07**% of transaction data is missing.

The identity data also has a lot of missing data which are marked as NaN (null) values. A simple python code provided in iPython notebook to list out the number of null values from each column and **35.58**% of identity data is missing.

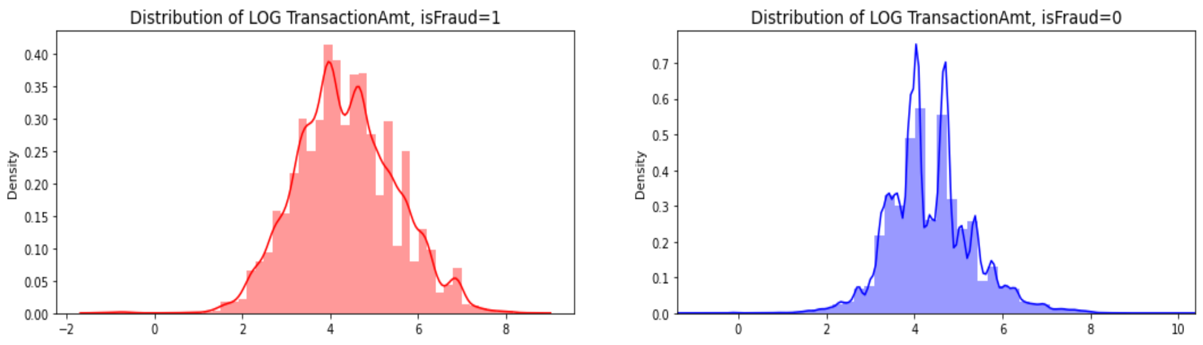
Only **24.42**% of Transaction data has the corresponding identity data.
https://github.com/dineshkb4u/ieee-fraud-detection/blob/master/Notebooks/Phase_1_Code.ipynb

Transaction data summary of maximum, minimum, mean, median, mode, standard deviation and interquartile ranges are listed in the iPython notebook. One of the interesting fields is the "Transaction Amount" from Transaction data. A simple distribution of transaction amount is as follows.



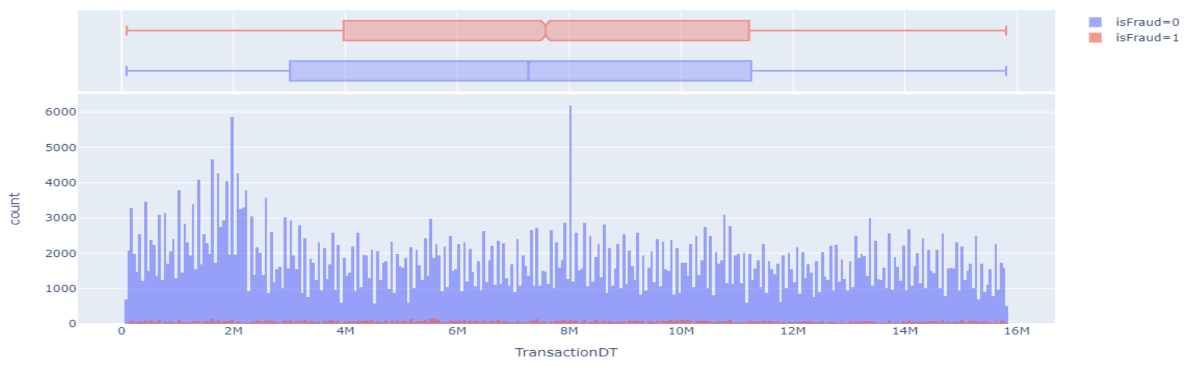
Transaction Amount Distribution

From the above plot, it clearly shows that the data has a lot of small amounts of transactions when compared to large amounts. Hence let us have a look at the log scale of this data.



Distribution of Log of Transaction Amounts

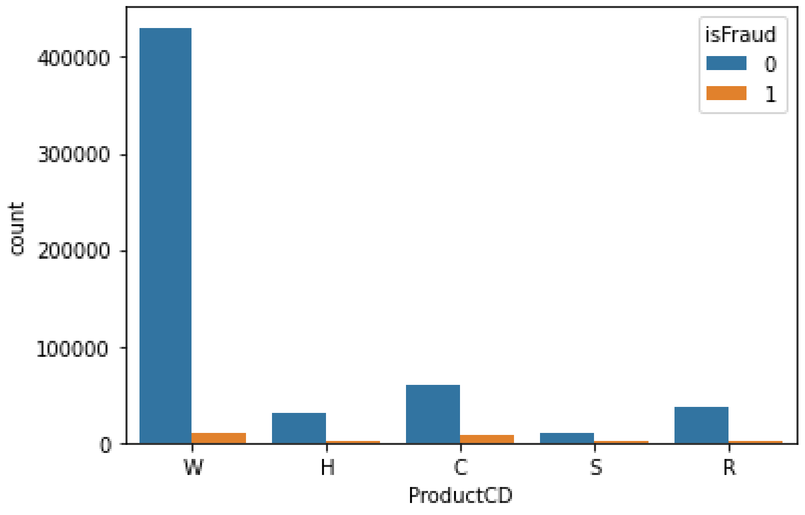
Now, let us see the distribution of transactions with respect to the Transaction Date column.



Distribution of Transactions over Time

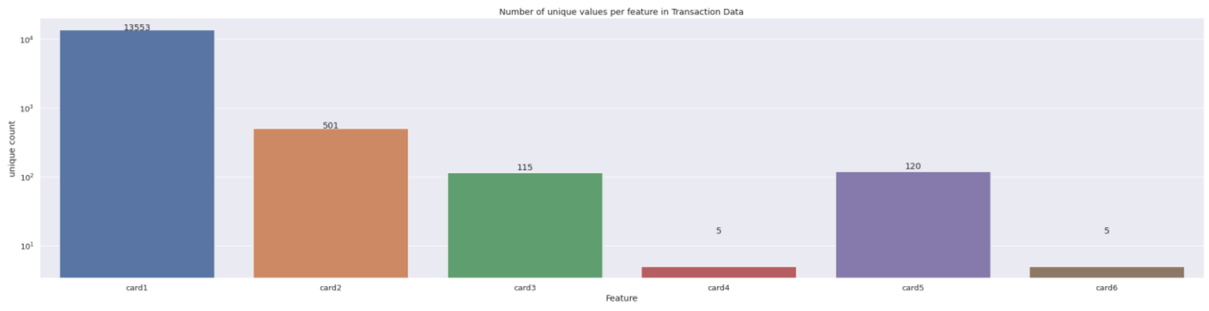
From the above graph the normal and fraud transactions are observed in all the timeframes and there are no specific time frame patterns observed for fraud transactions.

The next categorical feature is the Product Code and below the distribution of the same.



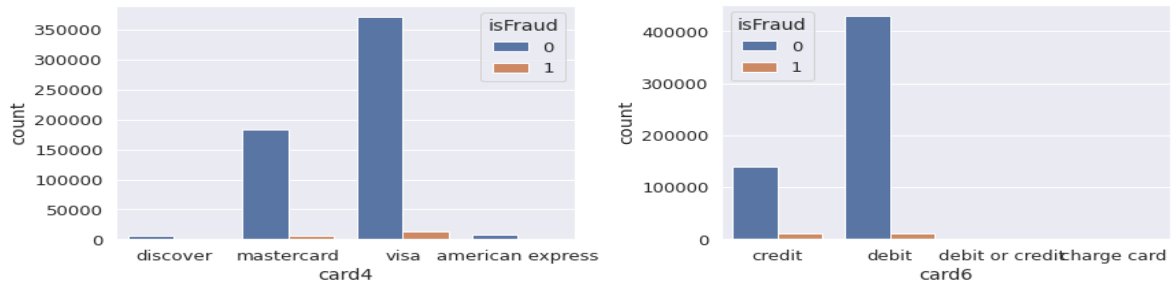
Distribution of Product Code for Normal & Fraud Transactions

Now let us see the data from the features named as card. There are 6 different categories present in the transaction data. The unique values present in each card1 to card6 features are as mentioned below.



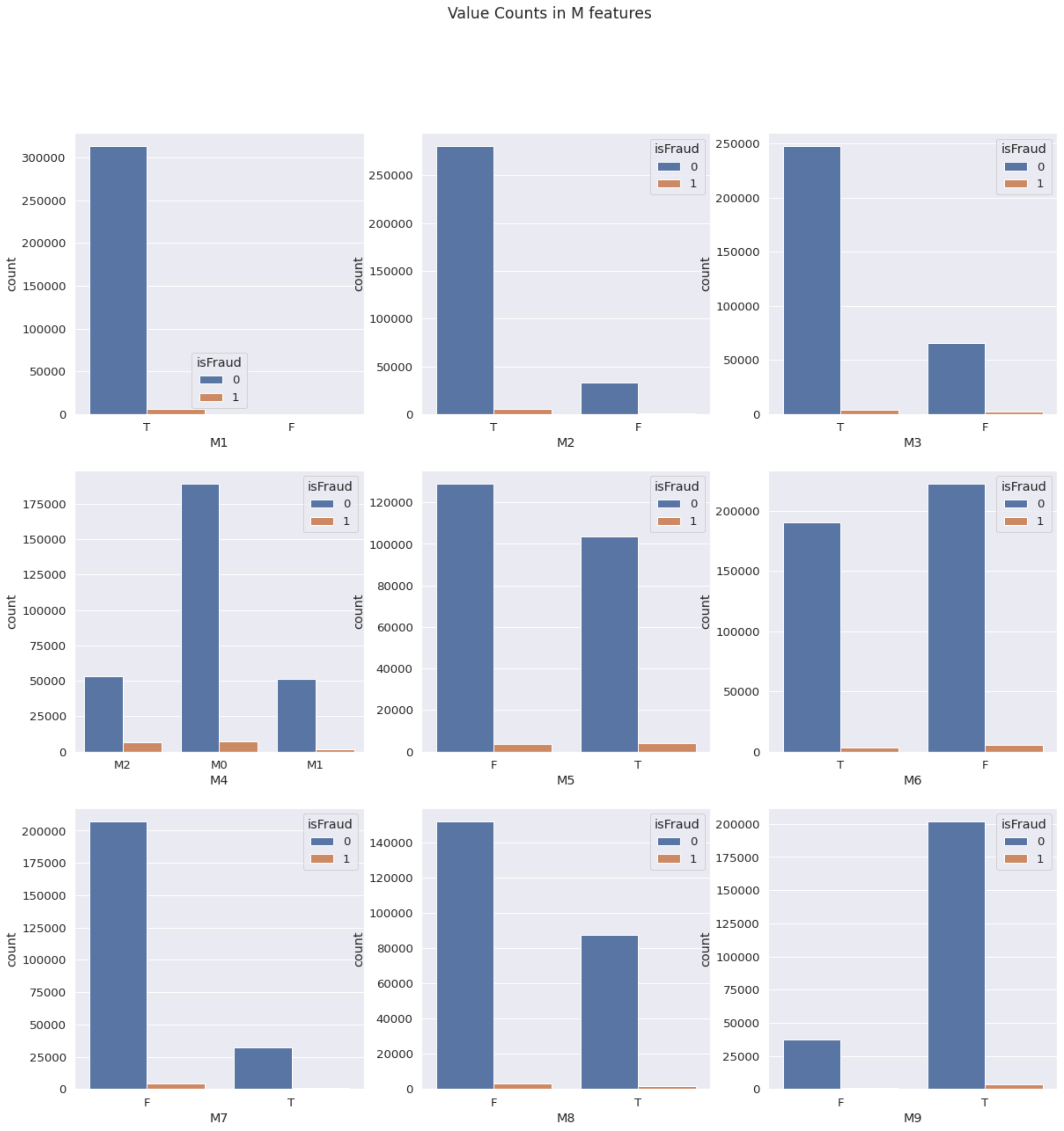
Number of unique values per feature in Transaction Data

From the above plot, we can observe that the card4 and card6 has very less unique features. Now lets see the normal and fraud transactions distribution from these 2 features.



Normal & Fraud transactions distribution for Card4 & Card6

Now let us check the distribution of M features from transaction data which are termed as categorical features in the Kaggle data.

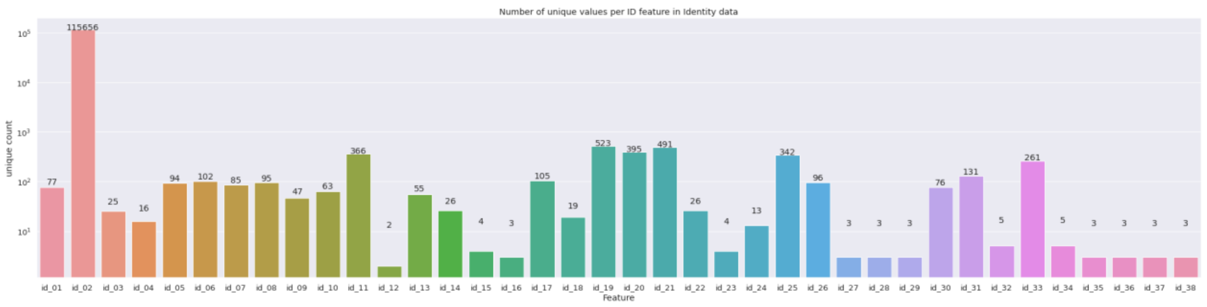


Distribution of Normal & Fraud transactions for M features

The columns addr1, addr2, dist1, dist2 have numerals and P_emaildomain, R_emaildomain have email domains. The unique values are printed in the python code.

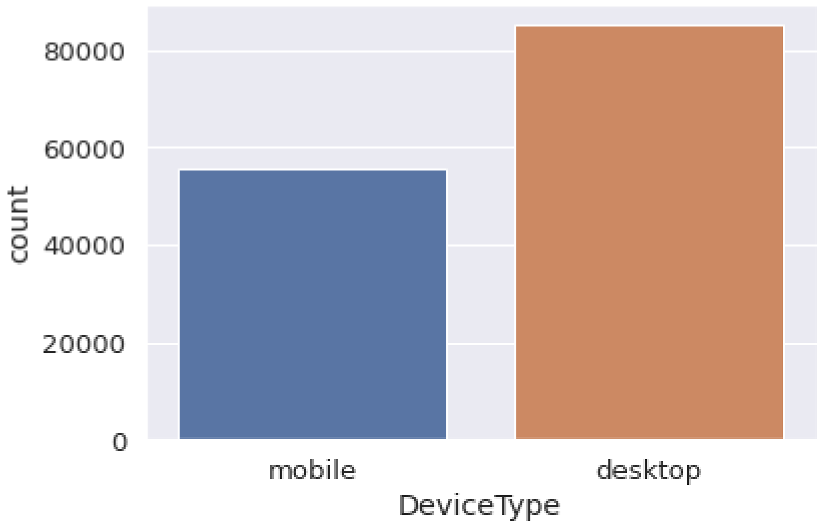
There are many features provided in transaction data. The features named as C1 - V339 are the engineered features generated/designed by the Vesta corporation and provided as part of Kaggle challenge.

The unique values across Id features from Identity data as mentioned below.



Unique values in Id features from Identity data

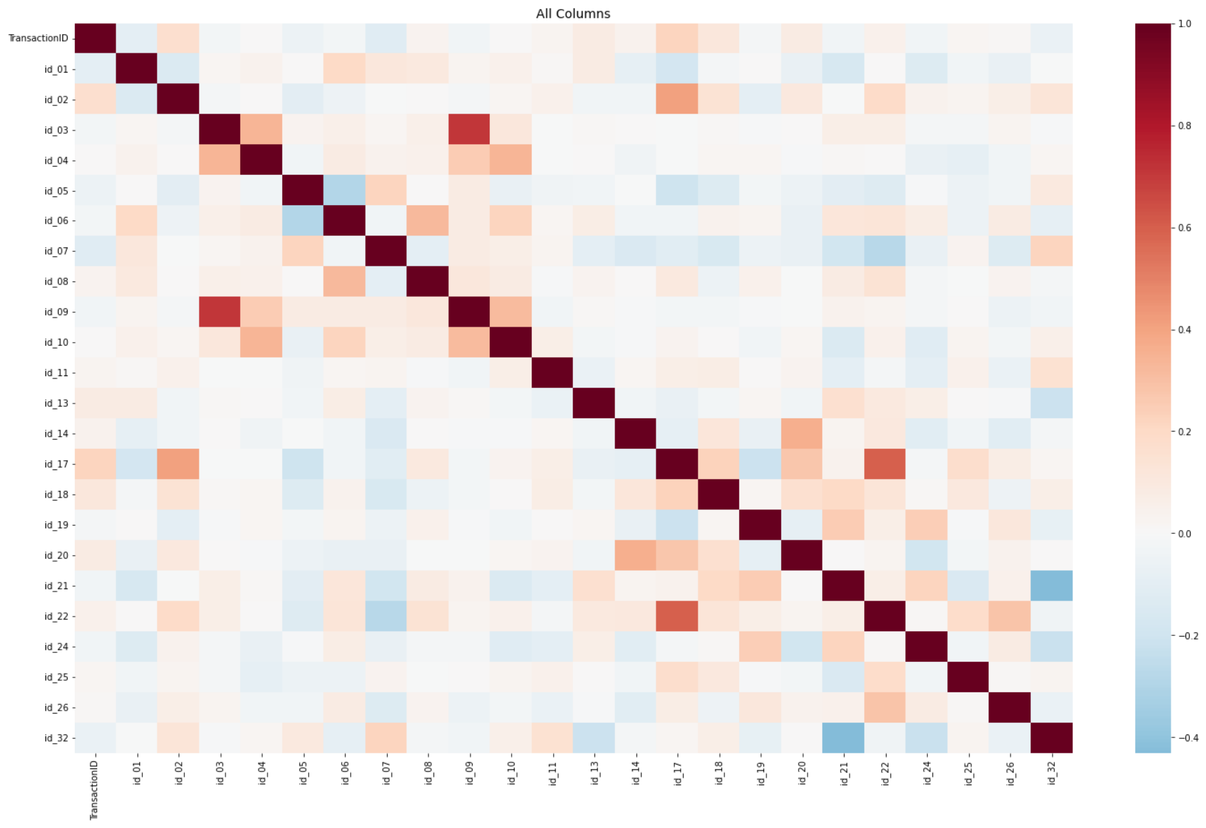
The distribution of device type mobile or desktop from identity data is as follows:



Distribution of DeviceType form Identity data

There are many unique values present for Device Info from identity data. This feature captures the device details like device model/company, operating system, etc.

The correlation between the features in identity data is as follows.



Correlation between the feature in Identity data

From the above heat map, we can observe that there are few features in Identity data which are highly correlated.

Feature Engineering/Encoding

Find the columns where the data provided in a column/feature was 90% or more of the given data then these columns can be dropped and test the model for better performance.

Fill the NaN values using the pandas fillNa function with a global value like “-999” as discussed in the query session with mentor and test the model accordingly.

As the data is highly imbalanced either re-sample(over sample and under sample) or generate the synthetic data points using methods like SMOTE.

Encode the string columns into numerical vectors using methods like Tf-Idf or W2V etc.

Based on the correlation heat map of transaction data it clearly shows that a lot of V columns are highly correlated. Hence by doing the dimensionality reduction methods some of these correlated V features can be removed.

Combining some features in transaction data can give better results which need to be tested by running multiple runs to find the performance improvement of the model and the features.

Both the transaction data and identity data needs to be combined and then need to encode the features like 1 hot encoding or concatenating a few of them to create new features.

It seems the D columns from transaction data have a relationship with hour of the transaction time based on the observations. Hence D columns can be engineered or refactored based on the Hr of Transaction and test the model.

Based on initial model performance, there can be an improvement based on the new features generated/generated from the existing features. This analysis can be done as part of Modelling.

Key Observations:

- The data was highly imbalanced in terms of normal and fraud transactions.
- **41.07**% of transaction data is missing or has NaN values.
- **35.58**% of identity data is missing or has NaN values.
- Only **24.42**% of Transaction data has the corresponding identity data.
- The transaction amounts are nearly normally distributed on the log scale.
- The normal and fraud transactions are observed in all the timeframes and there are no specific time frame patterns observed for fraud transactions.
- There are 5 product codes available and in all types of product codes the normal and fraud transactions are observed.
- The features card1, card2, card3 and card5 have many unique values. But the card4 and card6 have very few unique values and there is no specific pattern observed between normal and fraud transactions.
- It was mentioned that M features are categorical features and the normal & fraud transactions observed for all the features.
- The features addr1, addr2, dist1, dist2 have numerals.
- The features P_emaildomain, R_emaildomain have email domains.
- In the identity data the feature id_31 has the browser information which was used for the transaction.
- There are some features in identity data which are highly correlated.

The code for exploratory data analysis can be referred to in the below github location.
https://github.com/dineshkb4u/ieee-fraud-detection/blob/master/Notebooks/Phase_2_Code.ipynb

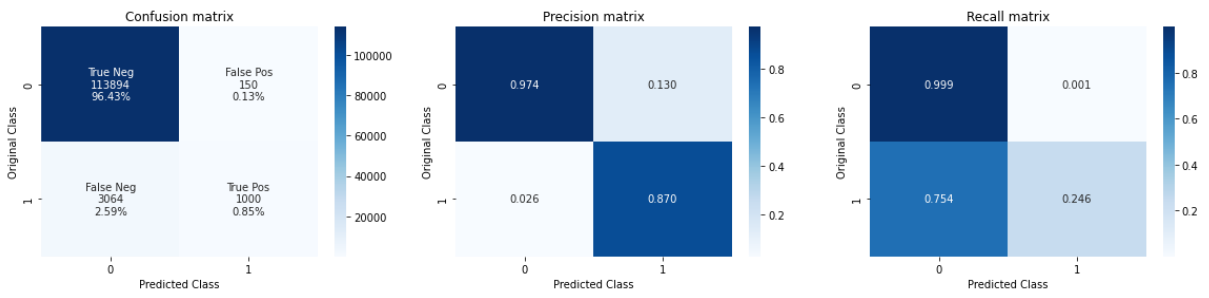
Base Modelling and Error Analysis:

The data used in this project is downloaded from the Kaggle competition. As explained earlier the data is broken into two files: identity and transaction. Not all transactions have corresponding identity information. Hence the data is merged using Transaction_ID with left join. Then the data was pre-processed using standard scalar fitting and also category values are fitted. Then the data has been used as is for training the below mentioned models as base line.

1. RandomForest classifier
2. XGBClassifier
3. Simple Neural Network model

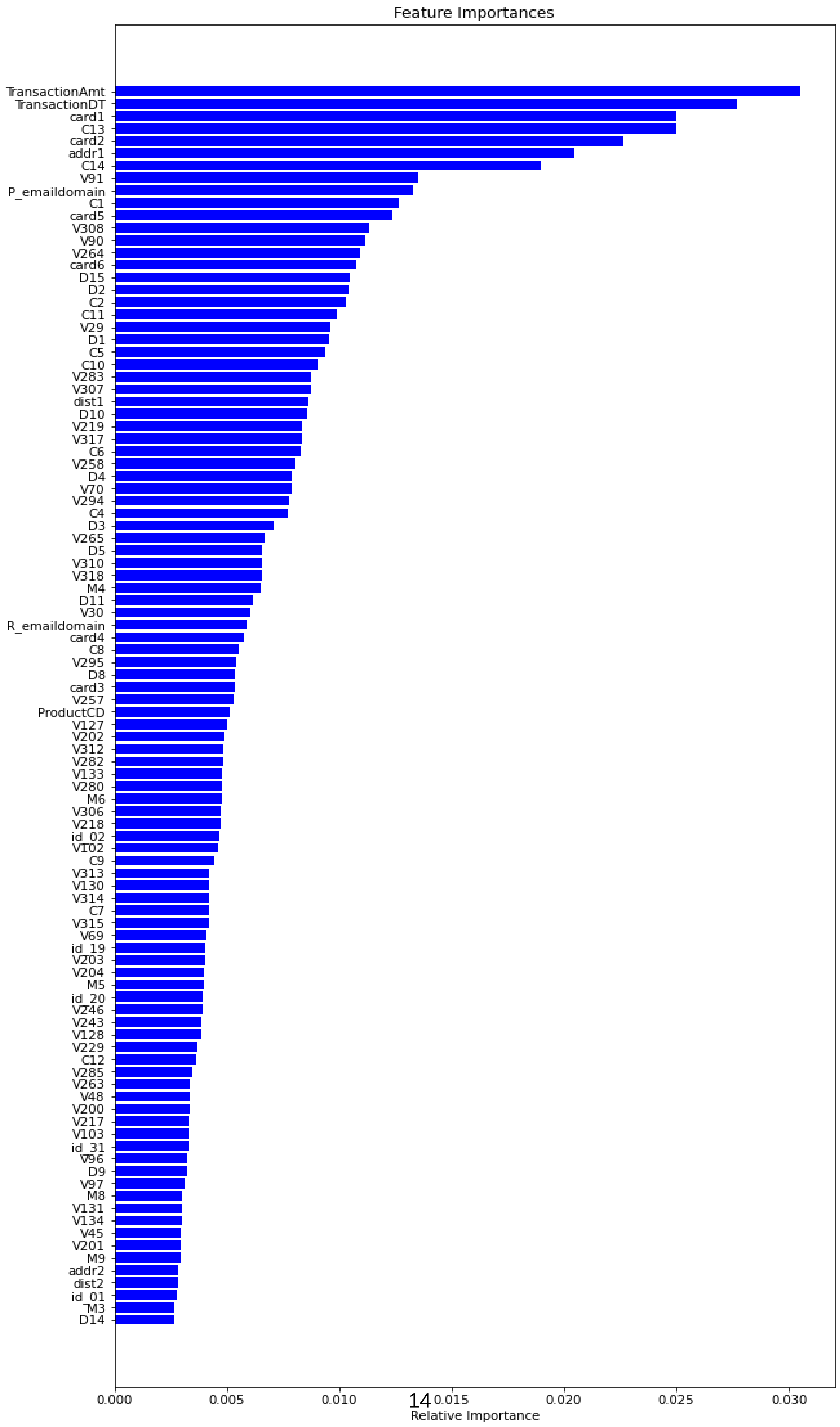
RandomForest Classifier:

The data has been splitted into 80% train data & 20% test data and then fed into the RandomForest classifier with k folds = 3. The average test data AUC-ROC score obtained is 0.7017917404949662 with all the default parameters used in the baseline run. The results with test data as a confusion matrix is shown below.



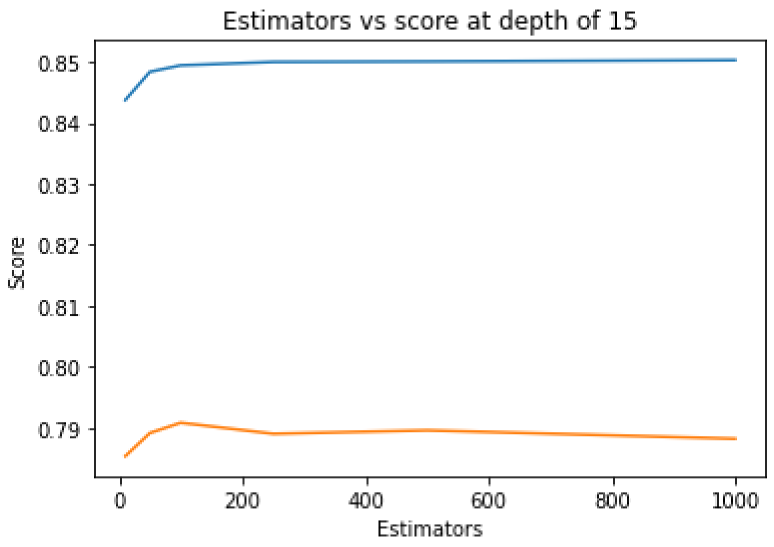
Test Data Confusion Matrix with RandomForest Classifier

From the above data the errors were observed more in the “False Negative” section and also the Recall % for “False Negative” is very high. The feature importance from the model for the first 100 features are listed in the image on the next page.


Feature Importances
TransactionAmt
TransactionDT
card1
C13
card2
addr1
C14
V91
P_emaildomain
C1
card5
V308
V90
V264
card6
D15
D2
C2
C11
V29
D1
C5
C10
V283
V307
dist1
D10
V219
V317
C6
V258
D4
V70
V294
C4
D3
V265
D5
V310
V318
M4
D11
V30
R_emaildomain
card4
C8
V295
D8
card3
V257
ProductCD
V127
V202
V312
V282
V133
V280
M6
V306
V218
id_02
V102
C9
V313
V130
V314
C7
V315
V69
id_19
V203
V204
M5
id_20
V246
V243
V128
V229
C12
V285
V263
V48
V200
V217
V103
id_31
V96
D9
V97
M8
V131
V134
V45
V201
M9
addr2
dist2
id_01
M3
D14
0.000
0.005
0.010
0.015
0.020
0.025
0.030
Relative Importance

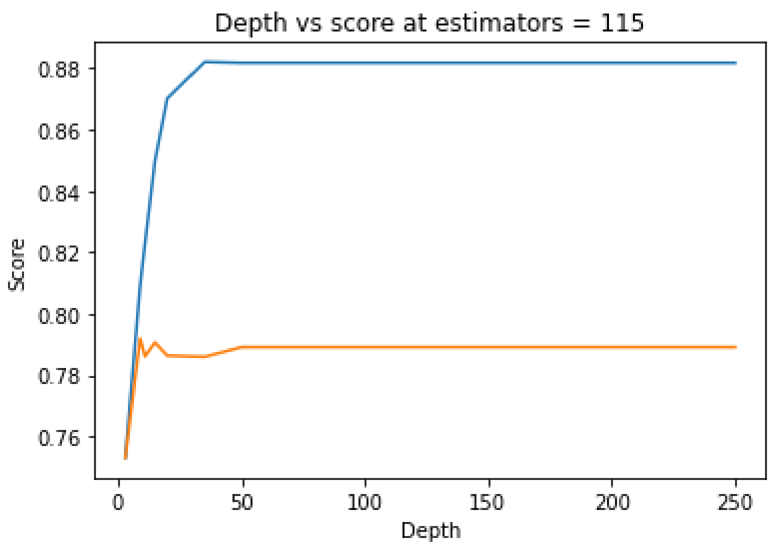
As part of hyper parameter tuning for RandomForrest classifiers a range of estimators were used and the AUC-ROC values were calculated.

```
Estimators =  10 Train Score 0.8437324130168552 test Score 0.7853479184508702
Estimators =  50 Train Score 0.8483543817956208 test Score 0.789141436327794
Estimators =  100 Train Score 0.8494057940886849 test Score 0.7908156502725707
Estimators =  250 Train Score 0.8499708620656027 test Score 0.7890088207079626
Estimators =  500 Train Score 0.8500421601067712 test Score 0.789575479328741
Estimators =  1000 Train Score 0.8502662634164415 test Score 0.7882007552272269
Text(0.5, 1.0, 'Estimators vs score at depth of 15')
```



From the above study the score was high near 100 estimators and hence the number of estimators used for the next study was fixed at 115. Now a range of depth parameter values are used to calculate the AUC-ROC score for the test data using fixed estimators. The results are as mentioned below.

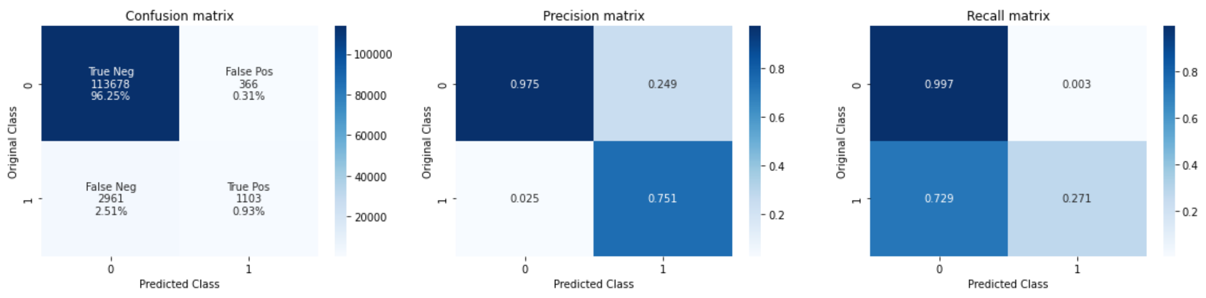
```
depth =  3 Train Score 0.7531123225705406 test Score 0.7530132295257226
depth =  9 Train Score 0.8090459190611452 test Score 0.7919924305013372
depth = 11 Train Score 0.8229322317428701 test Score 0.7861470772988018
depth = 15 Train Score 0.8496412879961707 test Score 0.7906970030492444
depth = 20 Train Score 0.8702092595654701 test Score 0.7863781232937583
depth = 35 Train Score 0.8821071496497312 test Score 0.7859989354847707
depth =  50 Train Score 0.881820268961144 test Score 0.7891618301003868
depth =  70 Train Score 0.881820268961144 test Score 0.7891618301003868
depth = 130 Train Score 0.881820268961144 test Score 0.7891618301003868
depth = 250 Train Score 0.881820268961144 test Score 0.7891618301003868
```



From the above study the score was high near depth at 15 and hence the depth was fixed at 15 for further. A complete study of hyper parameter tuning to get the best parameters will be studied in the next phase.

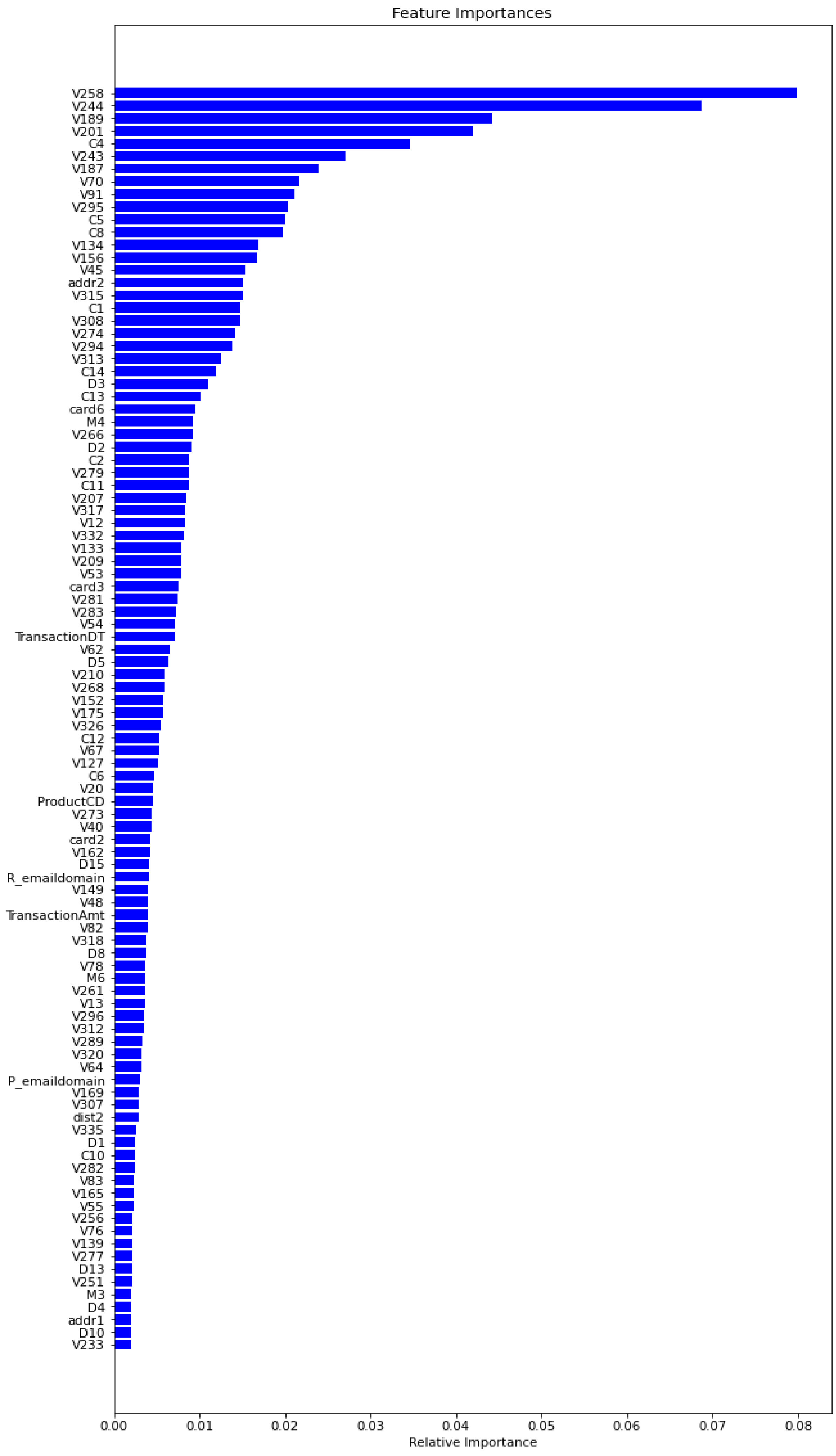
XGBClassifier:

The data has been splitted into 80% train data & 20% test data and then fed into the XGBclassifier with kfolds = 3. The average test data AUC-ROC score obtained is 0.6477976154172795 with all the default parameters used in the baseline run. The results with test data as a confusion matrix is shown below.



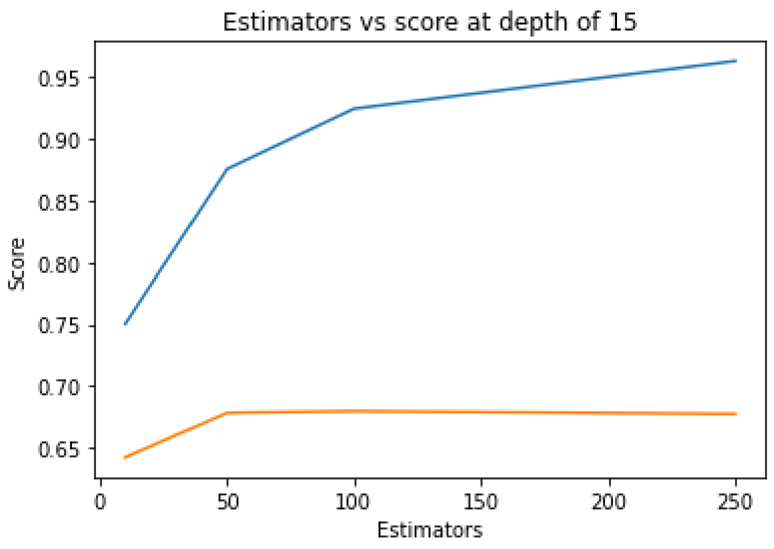
Test Data Confusion Matrix with XGBClassifier

From the above data the errors were observed more in the “False Negative” section and also the Recall % for “False Negative” is very high. The feature importance from the model for the first 100 features are listed in the image on the next page.


Feature Importances
V258
V244
V189
V201
C4
V243
V187
V70
V91
V295
C5
C8
V134
V156
V45
addr2
V315
C1
V308
V274
V294
V313
C14
D3
C13
card6
M4
V266
D2
C2
V279
C11
V207
V317
V12
V332
V133
V209
V53
card3
V281
V283
V54
TransactionDT
V62
D5
V210
V268
V152
V175
V326
C12
V67
V127
C6
V20
ProductCD
V273
V40
card2
V162
D15
R_emaildomain
V149
V48
TransactionAmt
V82
V318
D8
V78
M6
V261
V13
V296
V312
V289
V320
V64
P_emaildomain
V169
V307
dist2
V335
D1
C10
V282
V83
V165
V55
V256
V76
V139
V277
D13
V251
M3
D4
addr1
D10
V233
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
Relative Importance

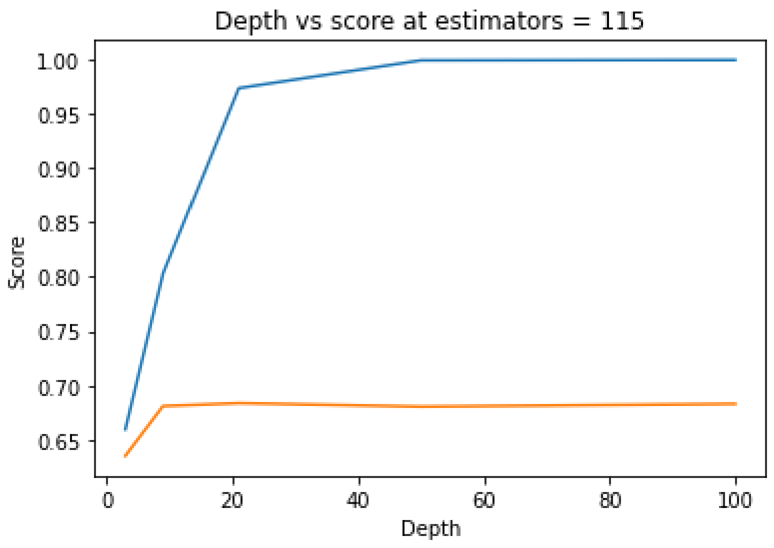
As part of hyper parameter tuning for XGBclassifier a range of estimators were used and the AUC-ROC values were calculated.

```
Estimators =  10 Train Score 0.7505373493753325 test Score 0.6426252467620592
Estimators =  50 Train Score 0.8754535852043711 test Score 0.6785748462328534
Estimators =  100 Train Score 0.9241635324693785 test Score 0.6799279208301148
Estimators =  250 Train Score 0.9627009186528406 test Score 0.6776865045456968
Text(0.5, 1.0, 'Estimators vs score at depth of 15')
```



From the above study the score was high near 100 estimators and hence the number of estimators used for the next study was fixed at 115. Now a range of depth parameter values are used to calculate the AUC-ROC score for the test data using fixed estimators. The results are as mentioned below.

```
depth =  3 Train Score 0.6593471157984903 test Score 0.6349203146347437
depth =  9 Train Score 0.8027347326420439 test Score 0.680837912021524
depth = 21 Train Score 0.9738226274278012 test Score 0.6835610545881311
depth = 50 Train Score 0.9995180432556179 test Score 0.6803586324742183
depth = 100 Train Score 0.999849388517381 test Score 0.6828889015622371
```



From the above study the score was high near depth at 15 and hence the depth was fixed at 15 for further. A complete study of hyper parameter tuning to get the best parameters will be studied in the next phase.

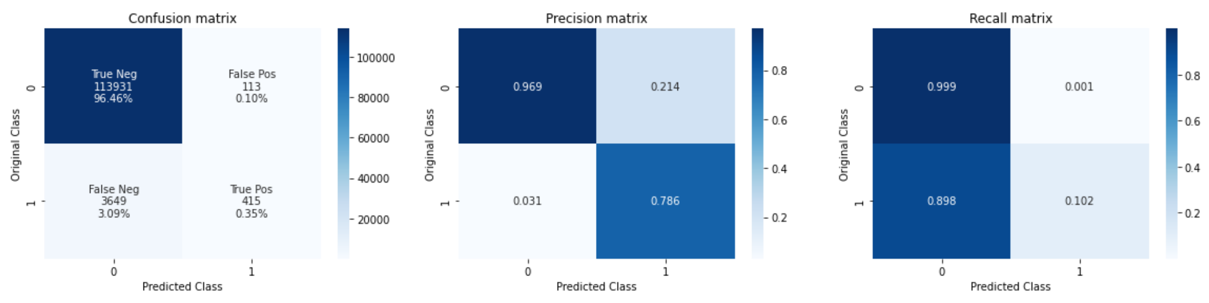
Simple Neural-Network Model:

The data has been splitted into 65% train data, 15% validation data & 20% test data and then fed into the simple Neural Network model. The average test data AUC-ROC score obtained is 0.7785 with all the default parameters used in the baseline run.

A sequential simple Neural Network model has been used as mentioned below.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)                (None, 16)               6912

 dropout (Dropout)            (None, 16)               0

 dense_1 (Dense)              (None, 8)                136

 dropout_1 (Dropout)          (None, 8)                0

 dense_2 (Dense)              (None, 4)                36

 dropout_2 (Dropout)          (None, 4)                0

 dense_3 (Dense)              (None, 1)                5

=================================================================
Total params: 7,089
Trainable params: 7,089
Non-trainable params: 0
_________________________________________________________________
```

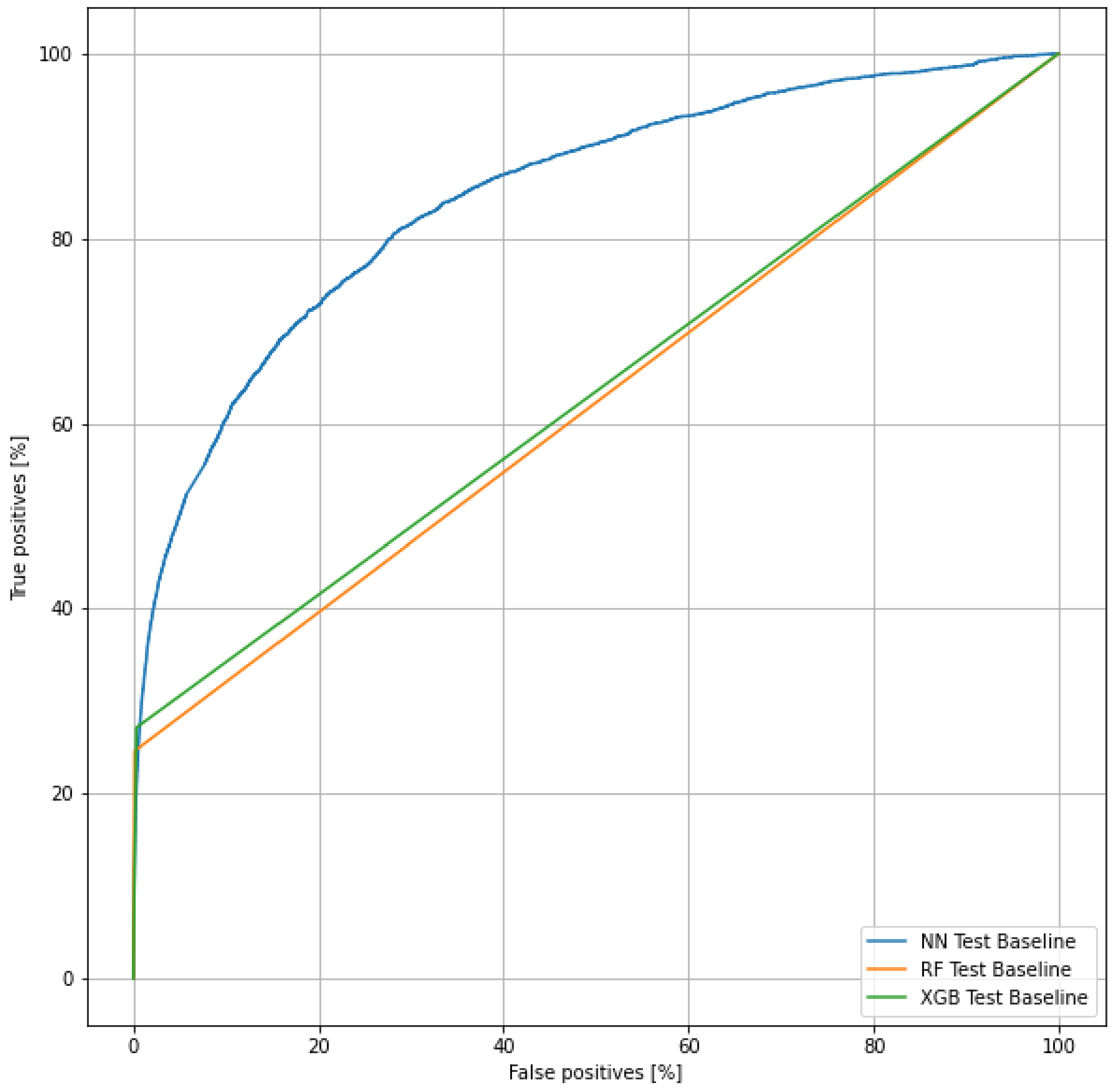
The results with test data as a confusion matrix is shown below.



Test Data Confusion Matrix with simple Neural Network Model

From the above data the errors were observed more in the “False Negative” section and also the Recall % for “False Negative” is very high.

All the above 3 models AUC_ROC curves are drawn in the below image to compare their performance as baseline models. It can be observed that all the 3 models can be finetuned more to get more accuracy by hyper parameter tuning and other feature engineering techniques.



The code for exploratory data analysis can be referred to in the below github location.
https://github.com/dineshkb4u/ieee-fraud-detection/blob/master/Notebooks/Phase_3_Code.ipynb

Advanced Modelling and Feature Engineering:

The data used in this project is downloaded from the Kaggle competition. As explained earlier the data is broken into two files: identity and transaction. Not all transactions have corresponding identity information. Hence the data is merged using Transaction_ID with left join. Then the data was pre-processed using standard scalar fitting and also category values are fitted. Then the data has been used as is for training the below mentioned models as base line.

Feature Engineering & Selection:

As part of feature engineering and feature selection, the features are grouped based on their missing percentage values. These grouped features for 'V' columns are passed through the correlation and sub-grouped based on their correlation >0.8. Then these groups are further reduced to find the column that has more unique values in the group of columns. This helped to reduce the features of 210 from 434 original features by selecting the most unique values in the correlated features. The python code for this can be referenced in the link below.
https://github.com/dineshkb4u/ieee-fraud-detection/blob/master/Notebooks/FE1_XGBoost.ipynb

Below mentioned models are used with the original & new features selected as above.

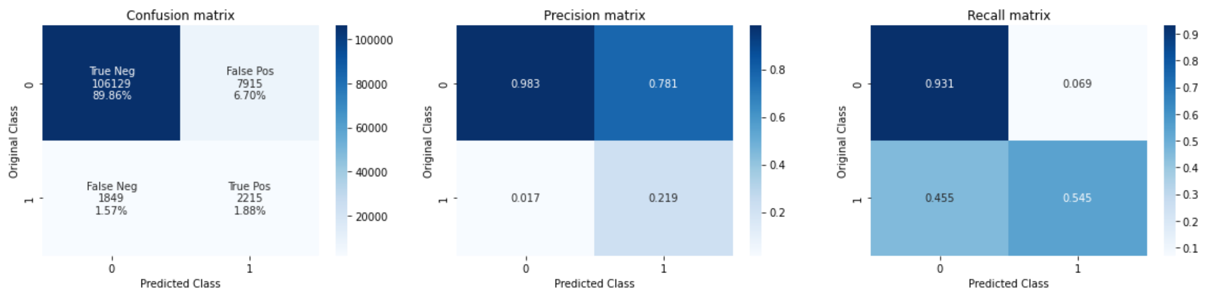
1. XGBClassifier
2. RandomForest classifier
3. Weighted Neural Network model

Models with Original Features:

The below models ran with the original features with no changes from feature engineering.

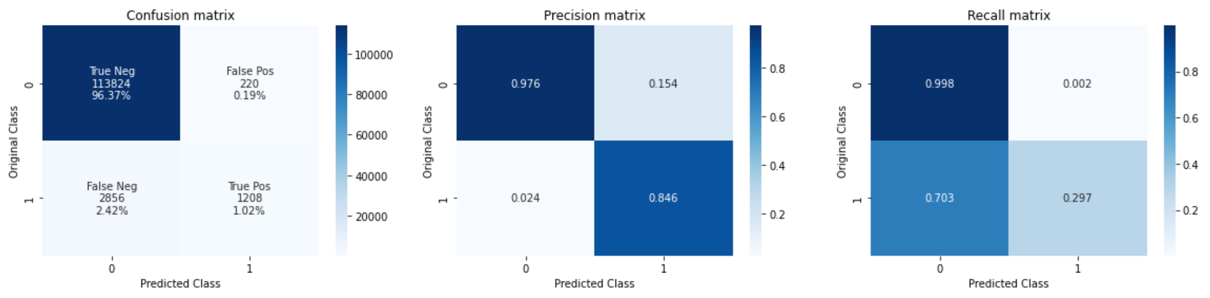
XGB Classifier:

The data has been splitted into 80% train data & 20% test data and then fed into the XGBclassifier and trained till there is no improvement in AUC-ROC score for next 100 iterations. The average test data AUC-ROC score obtained is 0.7378132450674515 with all the original features used in the run. The results with test data as a confusion matrix is shown below.



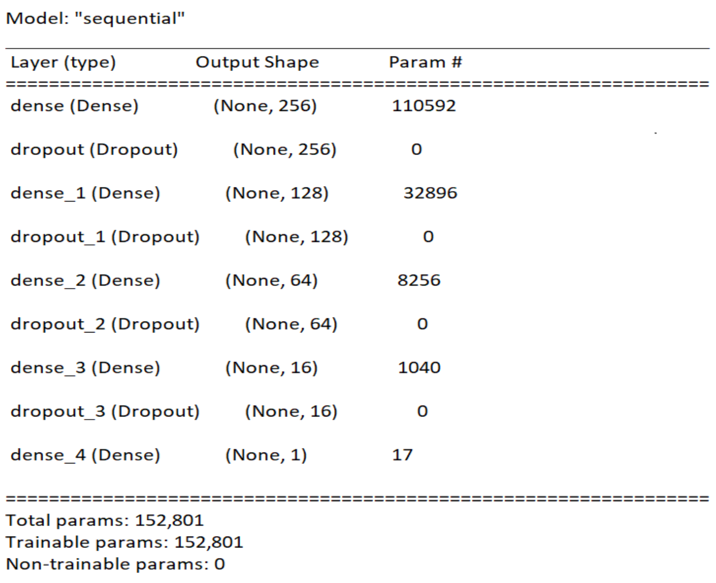
RandomForest Classifier:

The data has been splitted into 80% train data & 20% test data and then fed into the RandomForest classifier and trained till there is no improvement in AUC-ROC score for next 100 iterations. The average test data AUC-ROC score obtained is 0.6476575072419901 with all the original features used in the run. The results with test data as a confusion matrix is shown below.

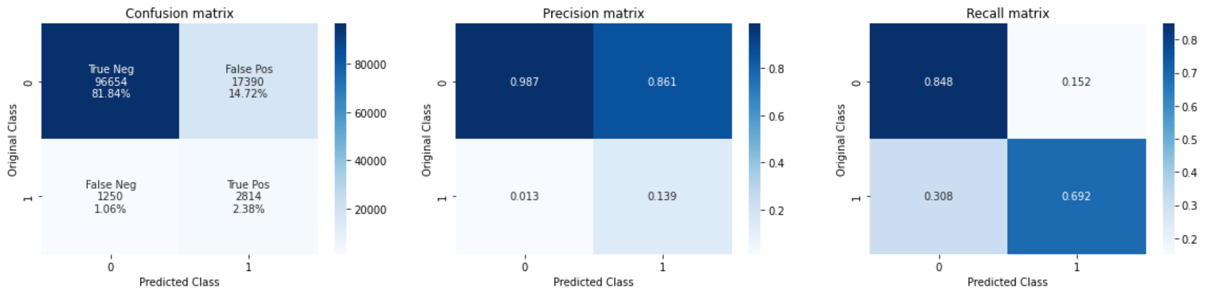


Weighted Neural Network Model:

The data has been splitted into 80% train data & 20% test data and then fed into the Weighted Neural Network Model and trained till there is no improvement in AUC-ROC score.


```
Model: "sequential"
_________________________________________________________________
Layer (type)            Output Shape          Param #
=================================================================
dense (Dense)             (None, 256)          110592

dropout (Dropout)          (None, 256)           0

dense_1 (Dense)           (None, 128)           32896

dropout_1 (Dropout)        (None, 128)           0

dense_2 (Dense)           (None, 64)           8256

dropout_2 (Dropout)        (None, 64)           0

dense_3 (Dense)           (None, 16)           1040

dropout_3 (Dropout)        (None, 16)           0

dense_4 (Dense)           (None, 1)          17

=================================================================
Total params: 152,801
Trainable params: 152,801
Non-trainable params: 0
```

The average test data AUC-ROC score obtained is 0.8446878790855408 with all the original features used in the run. The results with test data as a confusion matrix is shown below.

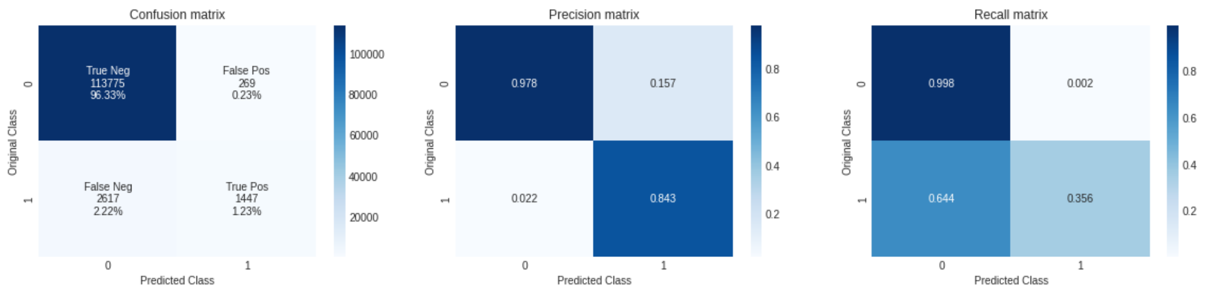


The python code for this can be referenced in the link below.
https://github.com/dineshkb4u/ieee-fraud-detection/blob/master/Notebooks/Phase_4_Code.ipynb

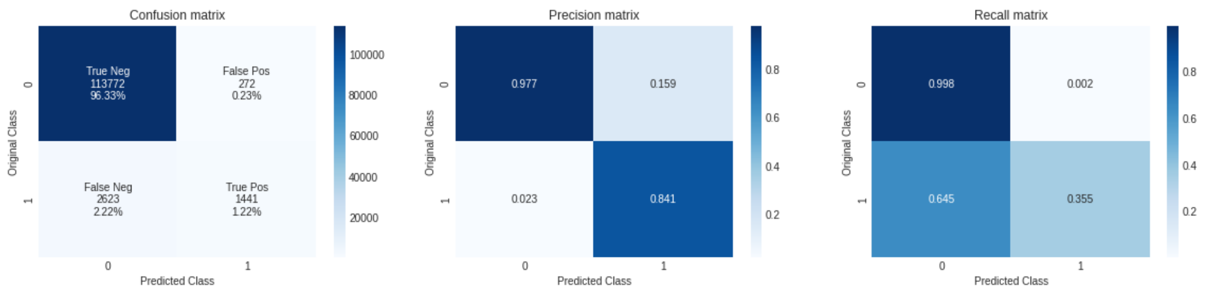
Models with Reduced Features:

XGB Classifier:

The data has been splitted into 80% train data & 20% test data and then fed into the XGBclassifier and trained till there is no improvement in AUC-ROC score for next 100 iterations. The average test data AUC-ROC score obtained is 0.67684720543694 with all the reduced features used in the run. The results with test data as a confusion matrix is shown below.



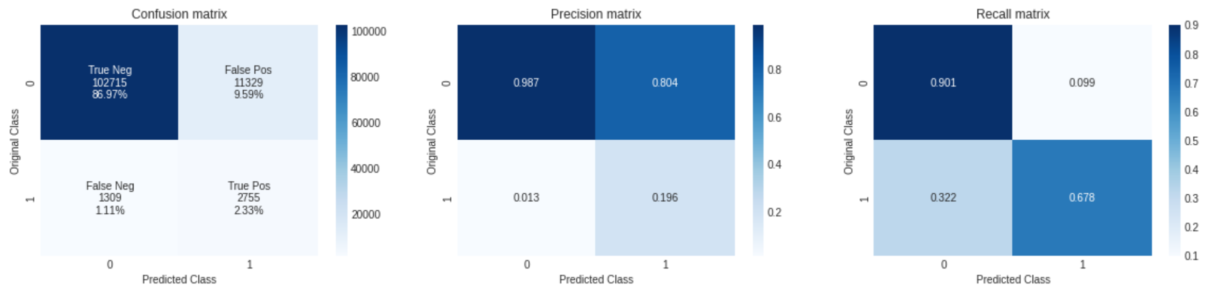
Another XGB classified model trained with reduced V features only and the average test data AUC-ROC score obtained is 0.6760958636423515 with all the reduced features used in the run. The results with test data as a confusion matrix is shown below.



The python code for this can be referenced in the link below.
https://github.com/dineshkb4u/ieee-fraud-detection/blob/master/Notebooks/FE2_XGBoost.ipynb
https://github.com/dineshkb4u/ieee-fraud-detection/blob/master/Notebooks/FE3_XGBoost.ipynb

RandomForest Classifier:

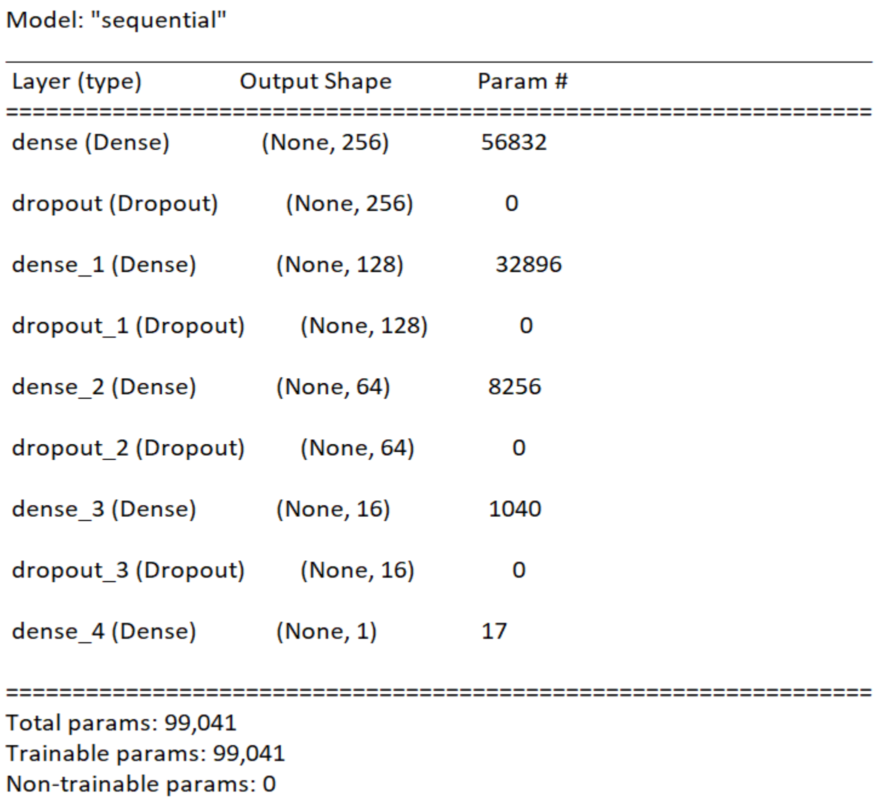
The data has been splitted into 80% train data & 20% test data and then fed into the RandomForest classifier and trained till there is no improvement in AUC-ROC score for next 100 iterations. The average test data AUC-ROC score obtained is 0.7892823458179009 with all the reduced features used in the run. The results with test data as a confusion matrix is shown below.



The python code for this can be referenced in the link below.
https://github.com/dineshkb4u/ieee-fraud-detection/blob/master/Notebooks/FE_RandomForest.ipynb

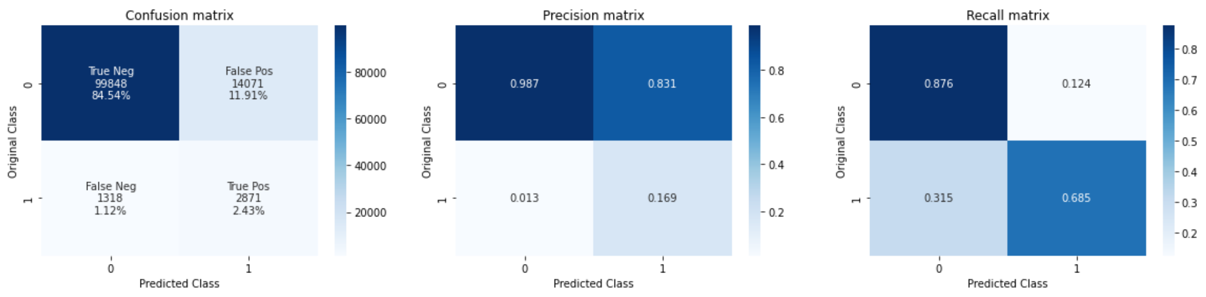
Weighted Neural Network Model:

The data has been splitted into 80% train data & 20% test data and then fed into the Weighted Neural Network Model and trained till there is no improvement in AUC-ROC score.



```
Model: "sequential"
_________________________________________________________________
Layer (type)             Output Shape          Param #
=================================================================
dense (Dense)              (None, 256)          56832

dropout (Dropout)            (None, 256)          0

dense_1 (Dense)             (None, 128)          32896

dropout_1 (Dropout)          (None, 128)          0

dense_2 (Dense)             (None, 64)           8256

dropout_2 (Dropout)          (None, 64)           0

dense_3 (Dense)             (None, 16)           1040

dropout_3 (Dropout)          (None, 16)           0

dense_4 (Dense)             (None, 1)            17

=================================================================
Total params: 99,041
Trainable params: 99,041
Non-trainable params: 0
```

The average test data AUC-ROC score obtained is 0.8651365041732788 with all the reduced features used in the run. The results with test data as a confusion matrix is shown below.



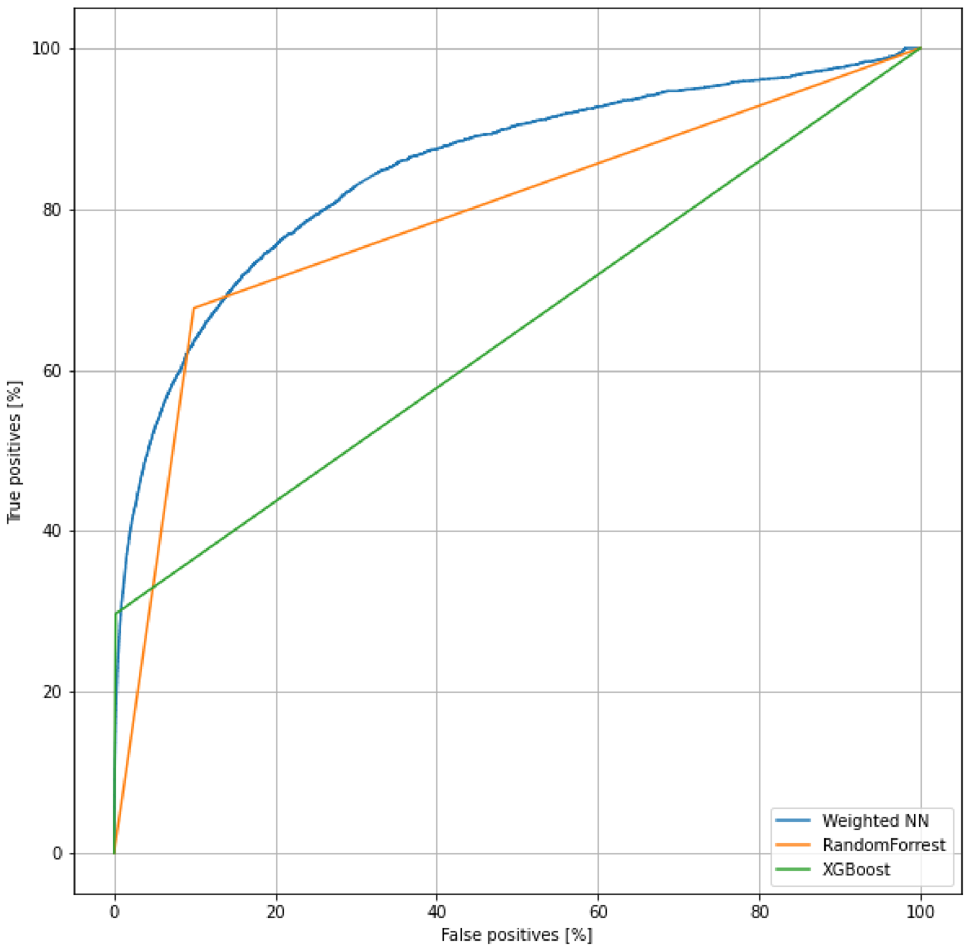
The python code for this can be referenced in the link below.
https://github.com/dineshkb4u/ieee-fraud-detection/blob/master/Notebooks/FE_WeightedMLP.ipynb

Comparison of all Models:

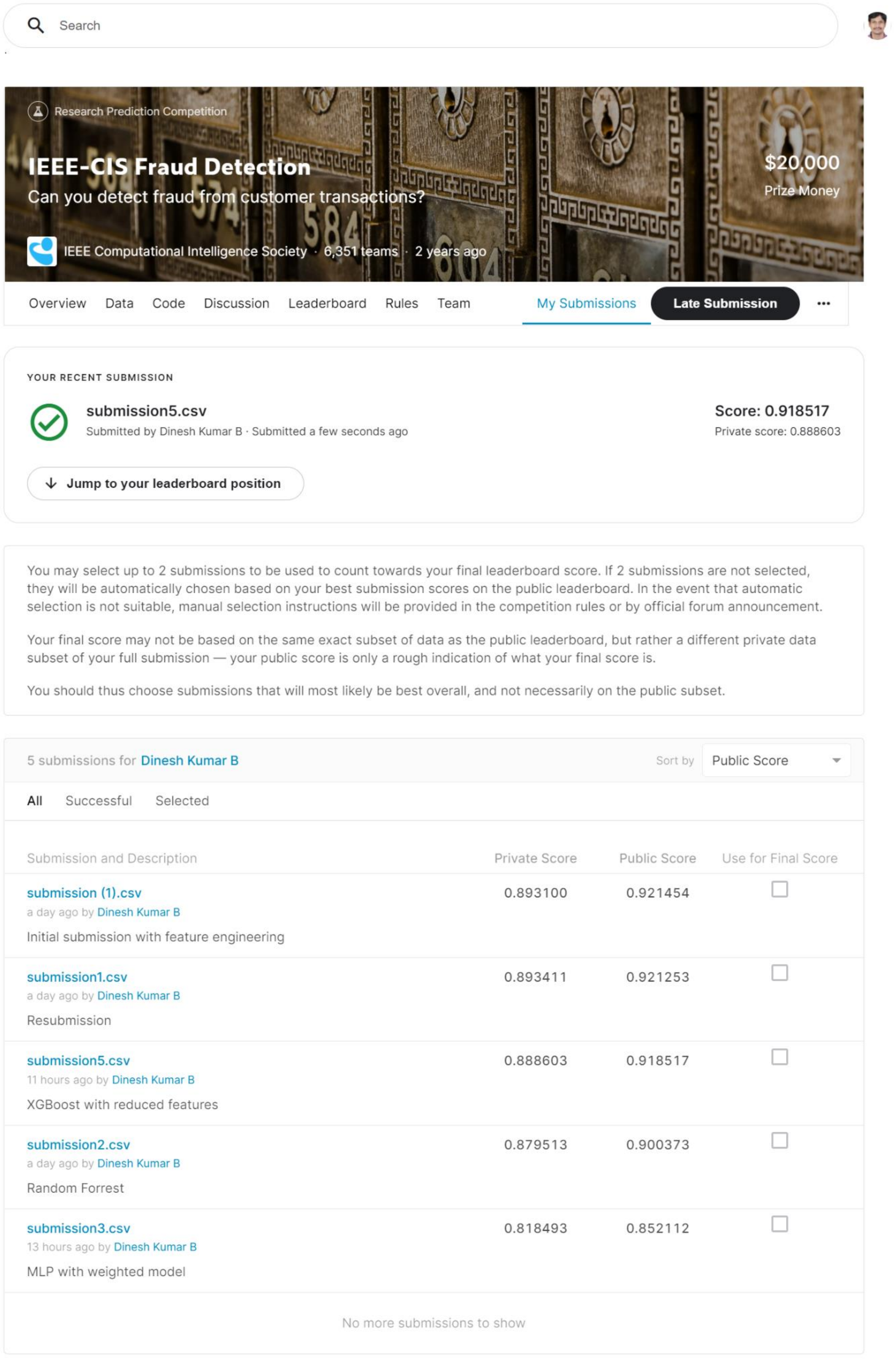
Below table has the summary of all the models used in this project are compared with their AUC-ROC score.

Model	AUC-ROC Score	Kaggle Test Score	TP	TN	FP	FN	FPR	FNR	Comments
Random Forrest	0.789048892	-	2751	102774	11270	1313	9.54%	1.11%	Takes less time and decent score
XGB Classifier	0.647657507	-	1208	113824	220	2856	0.19%	2.42%	Takes longer time and least score
Weighted Neural Network	0.844687879	-	2814	96654	17390	1250	14.72%	1.06%	Takes decent time and best score
Random Forrest (FE)	0.789282346	0.900373	2755	102715	11329	1309	9.59%	1.11%	Takes less time and decent score
XGB Classifier (FE)	0.676095864	0.921253	1441	113772	272	2623	0.23%	2.22%	Takes longer time and best Kaggle score
Weighted Neural Network (FE)	0.865136504	0.852112	2871	99848	14071	1318	11.91%	1.12%	Takes decent time and best score

Below graph shows the AUC-ROC curve for the models with all features.



Below is the Public & Private score from Kaggle competition for this project.



IEEE-CIS Fraud Detection

Can you detect fraud from customer transactions?

IEEE Computational Intelligence Society · 6,351 teams · 2 years ago

YOUR RECENT SUBMISSION

submission5.csv
Submitted by Dinesh Kumar B · Submitted a few seconds ago

Score: 0.918517
Private score: 0.888603

You may select up to 2 submissions to be used to count towards your final leaderboard score. If 2 submissions are not selected, they will be automatically chosen based on your best submission scores on the public leaderboard. In the event that automatic selection is not suitable, manual selection instructions will be provided in the competition rules or by official forum announcement.

Your final score may not be based on the same exact subset of data as the public leaderboard, but rather a different private data subset of your full submission — your public score is only a rough indication of what your final score is.

You should thus choose submissions that will most likely be best overall, and not necessarily on the public subset.

5 submissions for Dinesh Kumar B

Submission and Description	Private Score	Public Score	Use for Final Score
submission (1).csv a day ago by Dinesh Kumar B Initial submission with feature engineering	0.893100	0.921454	☐
submission1.csv a day ago by Dinesh Kumar B Resubmission	0.893411	0.921253	☐
submission5.csv 11 hours ago by Dinesh Kumar B XGBoost with reduced features	0.888603	0.918517	☐
submission2.csv a day ago by Dinesh Kumar B Random Forrest	0.879513	0.900373	☐
submission3.csv 13 hours ago by Dinesh Kumar B MLP with weighted model	0.818493	0.852112	☐

No more submissions to show

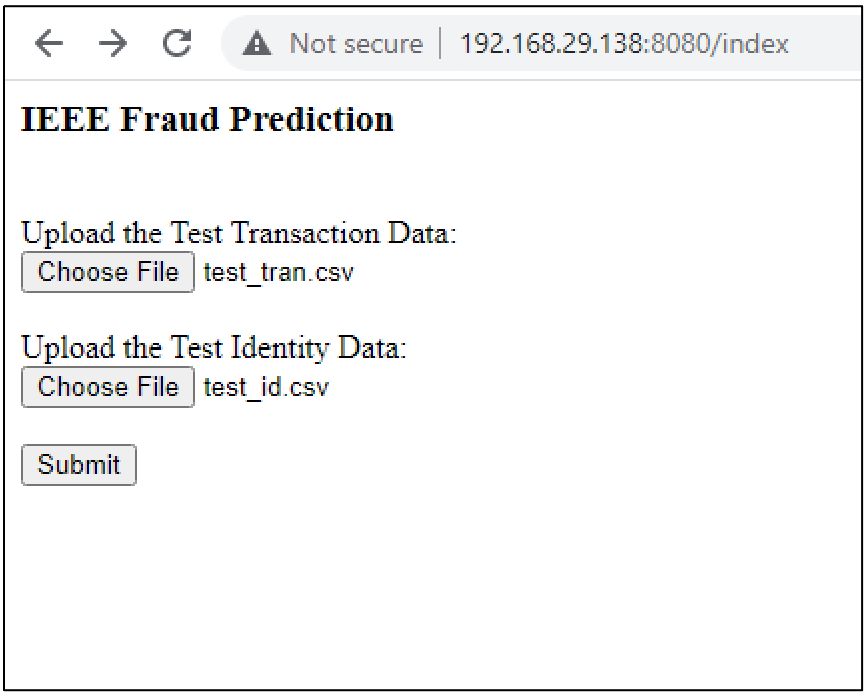
Model Deployment:

We chose the best XGBoost model with engineered features for deployment. We have followed the below deployment process.

1. Save the best model as a pickle file.
2. Create a module with all utility functions which can clean the raw input query, transform it using data pre-processing and predict the class of the query.
3. Create a main function for deployment which leverages the functions from the module.

I have deployed the model using in local & AWS platform.

The Test Data input html page from local running instance is shown below.



Not secure | 192.168.29.138:8080/index

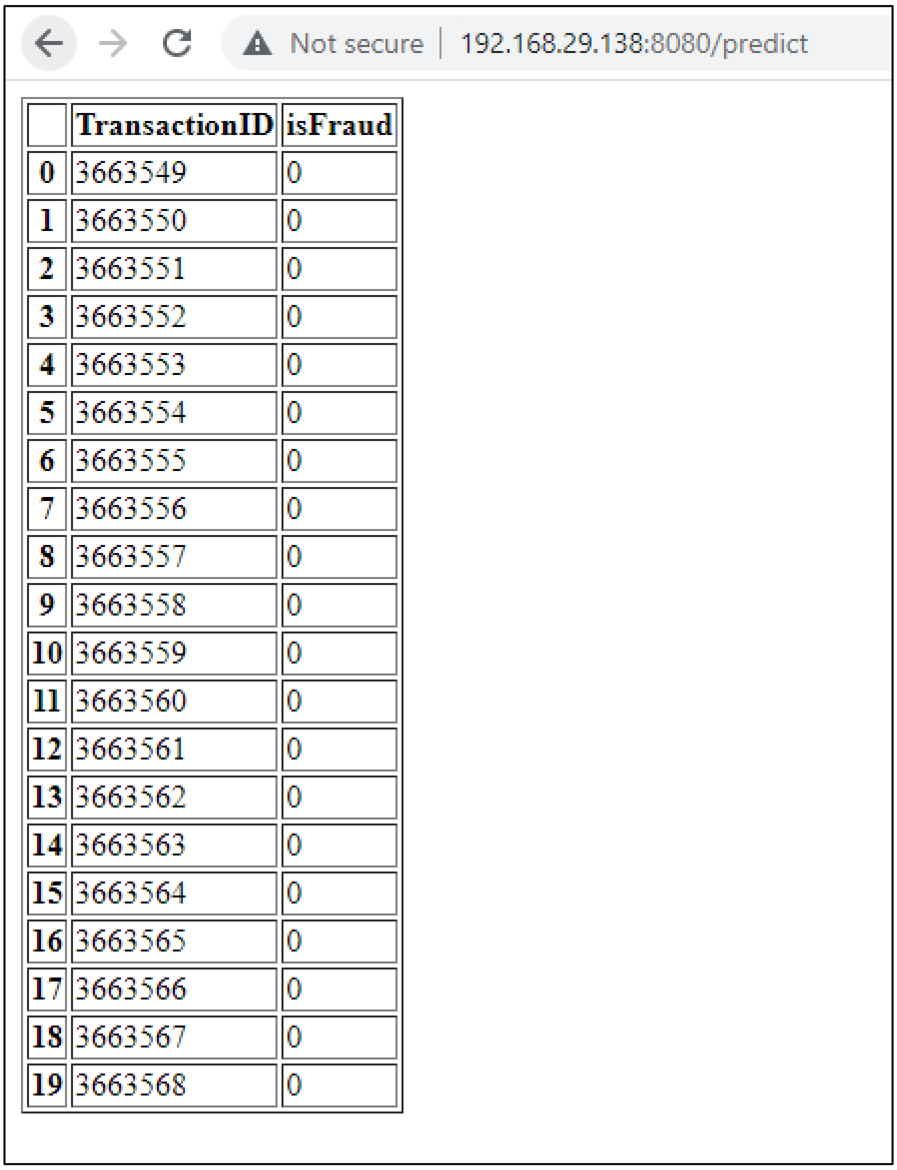
IEEE Fraud Prediction

Upload the Test Transaction Data:
Choose File test_tran.csv

Upload the Test Identity Data:
Choose File test_id.csv

Submit

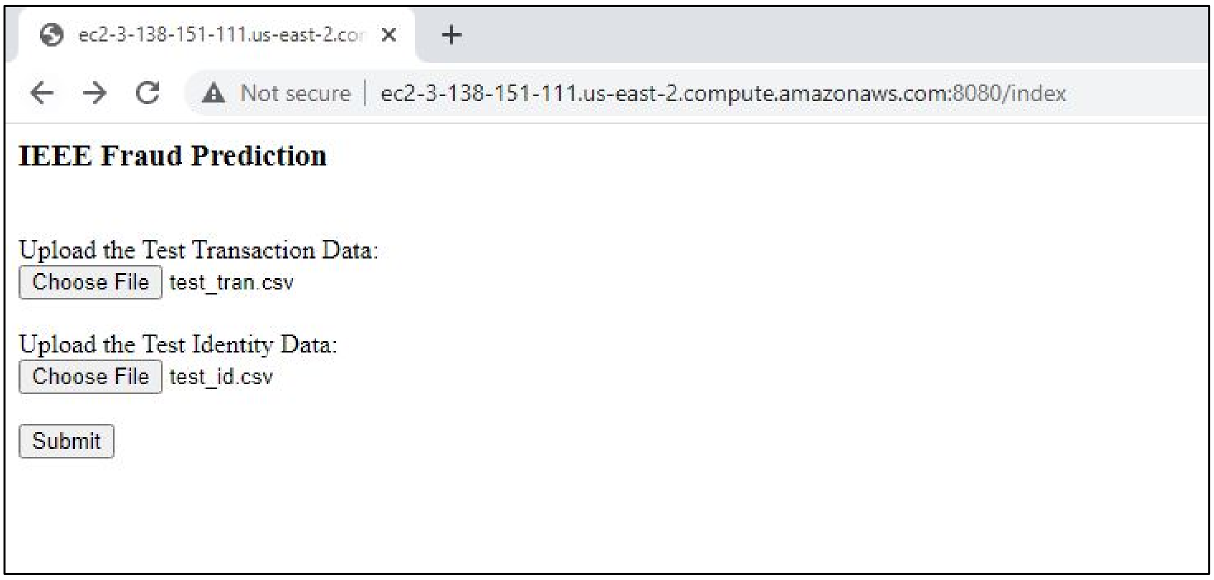
The Results output page from local running instance is shown below.



Not secure | 192.168.29.138:8080/predict

	TransactionID	isFraud
0	3663549	0
1	3663550	0
2	3663551	0
3	3663552	0
4	3663553	0
5	3663554	0
6	3663555	0
7	3663556	0
8	3663557	0
9	3663558	0
10	3663559	0
11	3663560	0
12	3663561	0
13	3663562	0
14	3663563	0
15	3663564	0
16	3663565	0
17	3663566	0
18	3663567	0
19	3663568	0

Below are the screenshots of the project application deployed from AWS RedHat Linux EC2 instance.

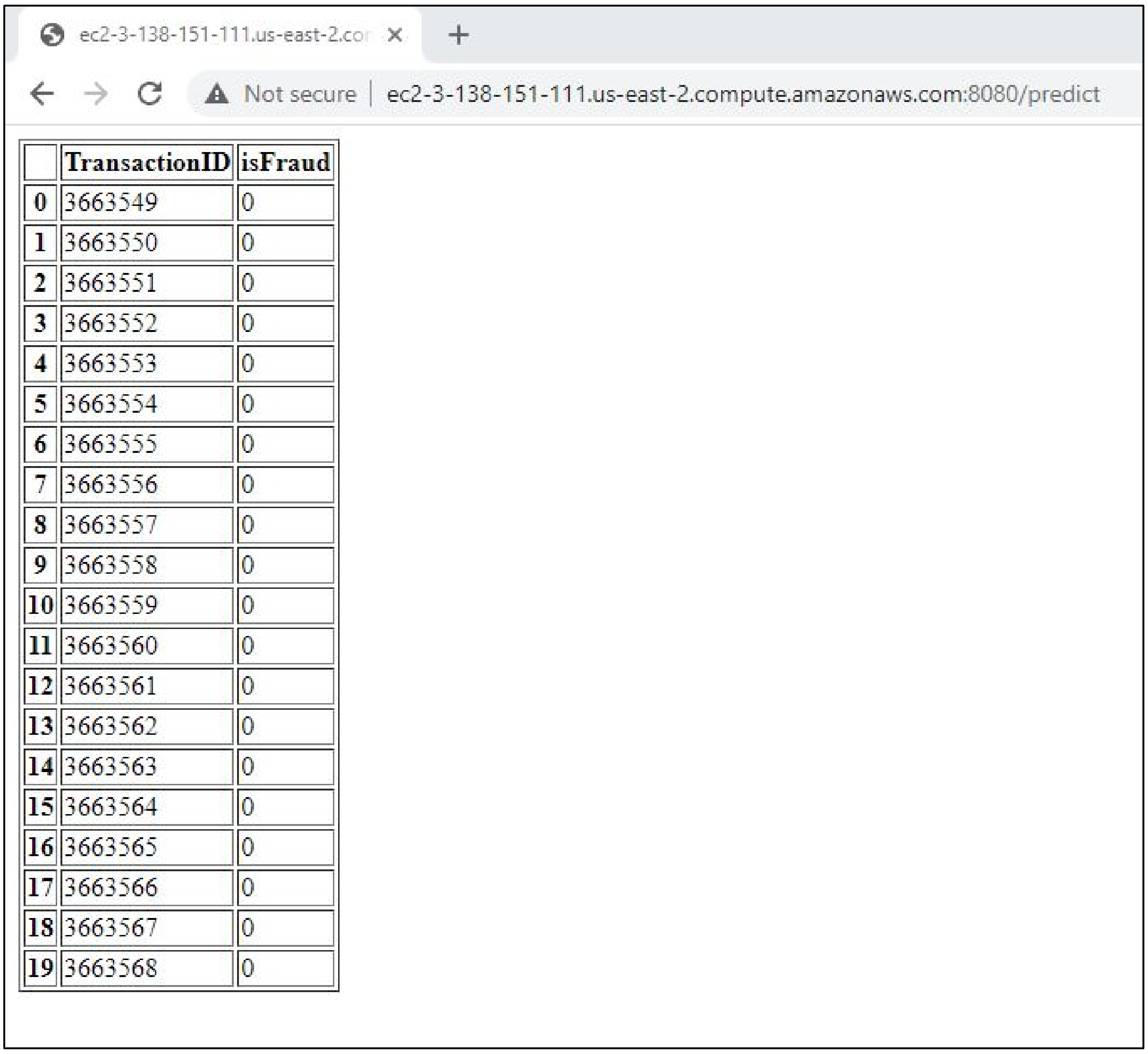

ec2-3-138-151-111.us-east-2.compute.amazonaws.com:8080/index

IEEE Fraud Prediction

Upload the Test Transaction Data:
Choose File test_tran.csv

Upload the Test Identity Data:
Choose File test_id.csv

Submit


ec2-3-138-151-111.us-east-2.compute.amazonaws.com:8080/predict

	TransactionID	isFraud
0	3663549	0
1	3663550	0
2	3663551	0
3	3663552	0
4	3663553	0
5	3663554	0
6	3663555	0
7	3663556	0
8	3663557	0
9	3663558	0
10	3663559	0
11	3663560	0
12	3663561	0
13	3663562	0
14	3663563	0
15	3663564	0
16	3663565	0
17	3663566	0
18	3663567	0
19	3663568	0

References:

1. https://arxiv.org/pdf/1505.01658.pdf
2. machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/
3. https://medium.com/@penggongting/understanding-roc-auc-pros-and-cons-why-is-bier-score-a-great-supplement-c7a0c976b679
4. https://medium.com/greyatom/lets-learn-about-auc-roc-curve-4a94b4d88152
5. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4356897/
6. http://www.iaeng.org/publication/IMECS2011/IMECS2011_pp442-447.pdf
7. medium.datadriveninvestor.com/rethinking-the-right-metrics-for-fraud-detection-4edfb629c423
8. towardsdatascience.com/precision-vs-recall-evaluating-model-performance-in-credit-card-fraud-detection-bb24958b2723
9. https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers#Single_metrics
10. https://en.wikipedia.org/wiki/Receiver_operating_characteristic
11. https://www.kaggle.com/c/ieee-fraud-detection/data
12. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/
13. machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/
14. https://medium.com/razorthink-ai/4-major-challenges-facing-fraud-detection-ways-to-resolve-them-using-machine-learning-cf6ed1b176dd
15. https://www.kaggle.com/paulrohan2020/performance-metrics-without-sklearn
16. https://medium.com/hackernoon/detecting-fraudulent-transactions-88031bac2382
17. https://towardsdatascience.com/using-machine-learning-to-detect-fraud-f204910389cf
18. https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
19. https://machinelearningmastery.com/logistic-regression-for-machine-learning/
20. https://towardsdatascience.com/log-loss-function-math-explained-5b83cd8d9c83
21. https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9100a3b2f
22. analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python
23. github.com/optuna/optuna-examples/blob/main/keras/keras_simple.py
24. youtube.com/watch?v=5nYqK-HaoKY
25. https://www.tensorflow.org/tutorials/structured_data/imbalanced_data#train_a_model_with_class_weights
26. https://towardsdatascience.com/ieee-cis-fraud-detection-top-5-solution-5488fc66e95f
27. https://kaggle.com/alijs1/ieee-transaction-columns-reference
28. https://www.kaggle.com/atenagm/ieee-fraud-detection-eda-ml
29. https://seaborn.pydata.org/tutorial/distributions.html
30. https://towardsdatascience.com/having-an-imbalanced-dataset-here-is-how-you-can-solve-it-1640568947eb

31. https://www.kaggle.com/nroman/eda-for-cis-fraud-detection
32. https://tensorflow.org/tutorials/structured_data/imbalanced_data
33. https://kaggle.com/alijs1/ieee-transaction-columns-reference
34. https://www.kaggle.com/atenagm/ieee-fraud-detection-eda-ml
35. https://www.tensorflow.org/tutorials/structured_data/imbalanced_data
36. https://www.kaggle.com/kyakovlev/ieee-cv-options
37. https://colab.research.google.com/drive/1HI7l4Bxi-Mi5TzeQmiShPDhs34F2LDFN#scrollTo=61TF-VLeHgmD
38. https://machinelearningmastery.com/bagging-and-random-forest-for-imbalanced-classification/
39. https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
40. https://www.youtube.com/watch?v=BPUfVq7RaY8
41. https://kaggle.com/alijs1/ieee-transaction-columns-reference