CREDIT CARD FRAUD DETECTION USING MACHINE LEARNING

BAN 502 01

Amna Mahmood (319631224)
Sbahat Riaz (319612305)
Shangool Riaz (319612304)
Zara Akhtar (319712229)

Table of Contents

1 Abstract

This report aims to evaluate the use of varying classification methods in conjunction with the management model CRISP-DM, to help in the prevention of bank fraud. Due to the immense growth of e-commerce and increased online-based payment possibilities, credit card fraud has become a deeply relevant global issue. Recently, there has been a major interest in applying machine learning algorithms as a data mining technique for credit card fraud detection. However, a number of challenges appear, such as lack of publicly available data sets, highly imbalanced class sizes, variant fraudulent behaviour and so on. In this report the performance of five machine learning algorithms: Random Forest, Decision Trees, Neural Networks and Gradient Boosting in detecting fraud on real-life data containing credit card transactions were compared. The problem of ever-changing fraud patterns is considered by employing incremental learning of selected Machine Learning algorithms in experiments. The performance of the techniques is evaluated based on commonly accepted metrics: accuracy score, precision score, recall score, F1 score, root mean square error (RMSE) and confusion matrix. In essence, the credit card fraud detection problem includes modelling past credit card transactions with the knowledge of the ones that turned out to be fraudulent. The model will then be used to identify whether a new transaction is fraudulent or not. The aim is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications.

2 Introduction

In today's ever-growing global society, the issue of committing fraud has become more relevant than ever. One of the areas that have experienced immense growth in e-commerce. Due to the increased number of possibilities in making payments of any kind and their easiness of use, e-commerce business establishments gained user confidence. Unfortunately, an increased number of users followed by increased revenue makes them vulnerable to fraudulent behaviour.

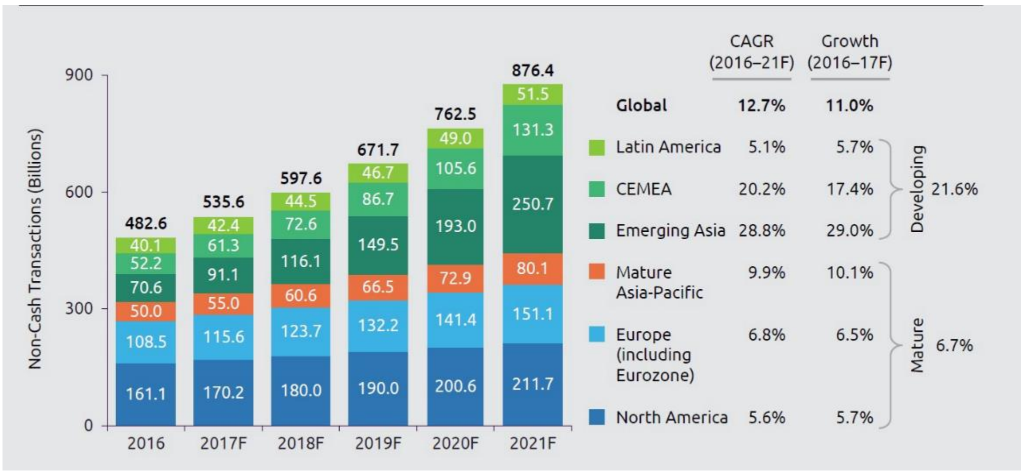


Figure 1: Number of Worldwide Non-Cash Transactions (Billions), by Region, 2016-2021F

According to the World Payments Report (Capgemini, 2018), in 2016 total non-cash transactions increased by 10.1% from 2015 for a total of 482.6 billion transactions! Despite credit card's ease of use, and popularity it comes with its fair share of drawbacks.

Most common card fraud types are application fraud, lost or stolen cards, account takeover, and counterfeit cards. The majority, of all fraudulent transactions made, are card-not-present (CNP) fraud. In the CNP scenario, the credit card information is obtained without the knowledge of cardholders and is used remotely in an attempt to commit fraud (T. P. Bhatla, V. Prabhu, A. Dua, 2003).

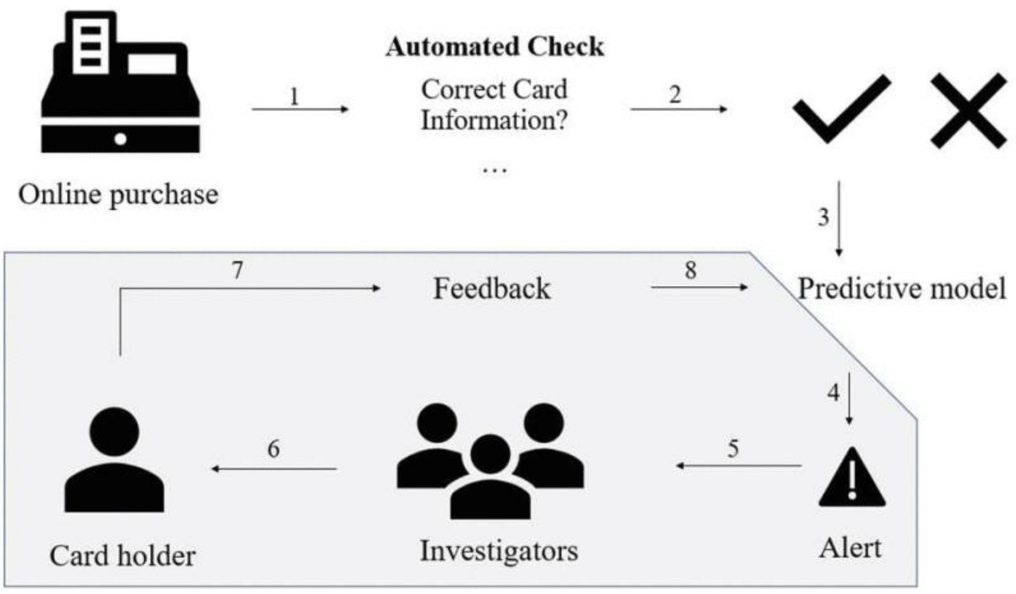


Figure:2 Typical Credit Card Fraud Detection (S.Bhattacharyya, 2011)

Fraud detection assumes identifying fraud as quickly as possible once it has been committed. Methods for detecting fraud are continuously adapting to ever-changing fraud strategies. Presently, fraud detection has been implemented by several methods such as statistics, rule engines, artificial intelligence, and data mining. As shown in Figure 2, the first level is automated card validation, which is performed in real-time, for the benefit of user satisfaction. If a card is not rejected, the transaction goes through the second level, denoted as the Predictive model. Throughout the past decade, there has been increased interest in applying machine learning as a data mining approach for the second level of credit card fraud detection (P. H. Tran, 2018)

Solving the fraud detection problems with the data mining technique boils out to classifying transactions to one of the two categories: fraud or not fraudulent. Classification is a data mining method that assigns items in a collection to target categories or classes. The goal of classification is to accurately predict the target class for each case in the data. The simplest type of classification problem is a binary classification where the target attribute has only two possible values. Credit card fraud detection is, in fact, a binary classification problem. This paper implements supervised machine learning algorithms for the classification of a credit-card transaction as either fraudulent or not-fraudulent (Data Mining Concepts).

3 CRISP – DM

The Cross-Industry Standard Process for Data Mining - CRISP-DM is a model if a data mining process used to solve problems by experts. The model identifies the different stages in implementing a data mining project.

3.1 Business Understanding

The aim of this project is to build a classifier that can detect credit card fraudulent transactions. A variety of machine learning algorithms will be deployed to discern fraudulent from non-fraudulent ones.

Due to the common use of credit cards, it has become that much easier for consumers to fall prey to fraudulent scams. Various types of credit card scams exist, however in general, it entails private information of consumers being stolen and credit cards being utilized by making unauthorized purchases.

Around 0.1% of all credit card transactions are fraudulent which equates to a huge financial loss. Some 12 billion credit card transactions were conducted in 2009, of which about 10 million were fraudulent. additionally, it was found that 0.04% of all accounts were active monthly were fraudulent. These proportions have not changed much over time.

Hence the project contains the following objectives.

- Visualize the dataset
- Implement algorithms such as Random Forest, Decision Trees, Gradient Boosting, Neural Networks, and Support Vector Machine
- Training, testing the data
- Evaluating the accuracy of the models
- Effectiveness of their implementation

Due to an imbalance in data, an algorithm that does not do any feature analysis and predicts all the transactions as non-frauds will also end up achieving an accuracy of 99.828%. Therefore, accuracy is not a correct measure of efficiency in this case. Hence other standards of correctness will be utilized while classifying transactions as fraud or non-fraud.

The metrics used will be:

- Precision
- Recall
- F1 Score
- The area under the ROC Curve (AUC)
- Root mean square error (RMSE)
- Confusion matrix

3.2 Data Understanding

3.2.1 Describing Data

Existing Dataset has been used from Kaggle, it was created by combining two data sources; the fraud transactions log file and all transactions log file. The Data set contains transactions made by European cardholders in September 2013. The transactions have occurred over a period of two days. It contains approximately 285 000 rows of data and 31 columns.

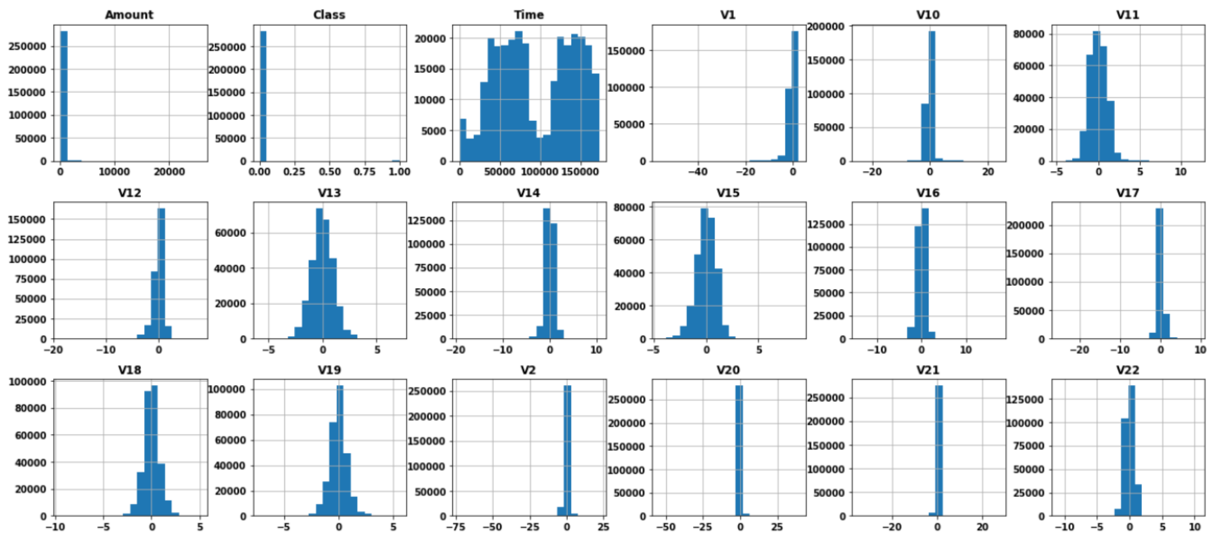
The fraud transactions log file holds all the online credit card fraud occurrences while all transactions log file holds all transactions stored by the corresponding bank within a specified time period. Due to the confidential disclosure agreement made between the bank and the authors of the paper, some of the sensitive attributes such as card number have been transformed using Principal Component Analysis (PCA) dimensionality reduction. While Time, Amount and Class have stayed in their original format.

The dataset contains 492 frauds out of 284,807 transactions, the frauds make up for 0.172% of all transactions, hence being highly imbalanced. The ‘Time’ column contains the seconds elapsed between each transaction from the first transaction. The ‘Class’ feature takes 1 for a fraudulent transaction and 0 for a non-fraudulent transaction, while the ‘V1-V28’ are variables containing the user’s identities and sensitive features, as the variables are transformed using PCA.

Attributes of the data source are as follows:

Field name:	Description:
Time	Number of seconds elapsed between the current transaction and the first transaction in the dataset
V1-V28	User identities and sensitive features have undergone Principal Component Analysis (PCA) Dimensionality function
Amount	Transaction amount
Class	1 for fraudulent transactions, 0 for otherwise

3.2.2 Exploring Data



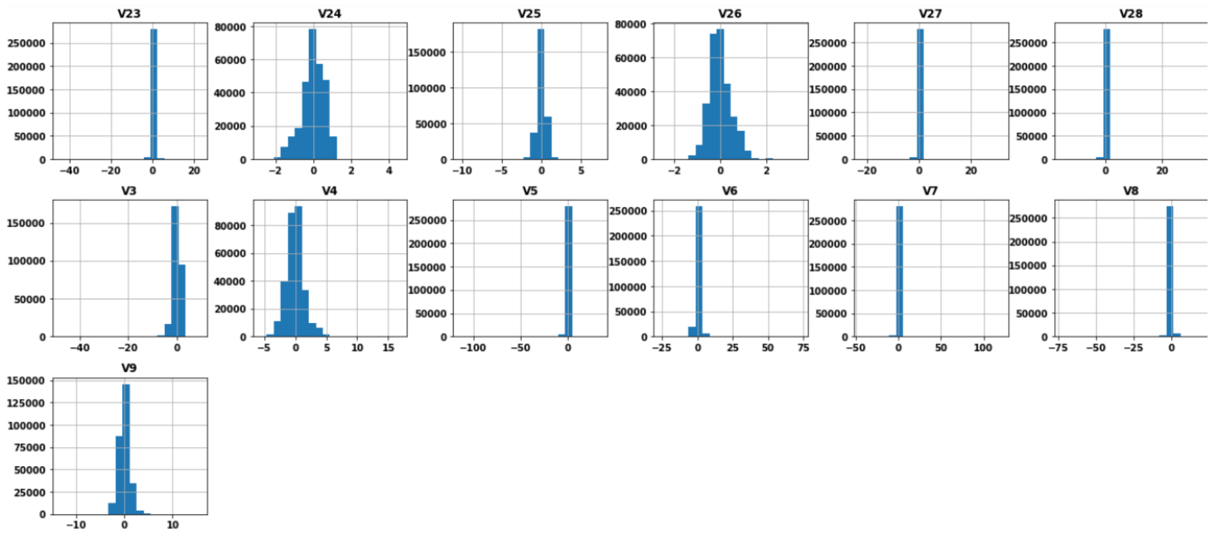


Figure 3: Histogram General Visualisation

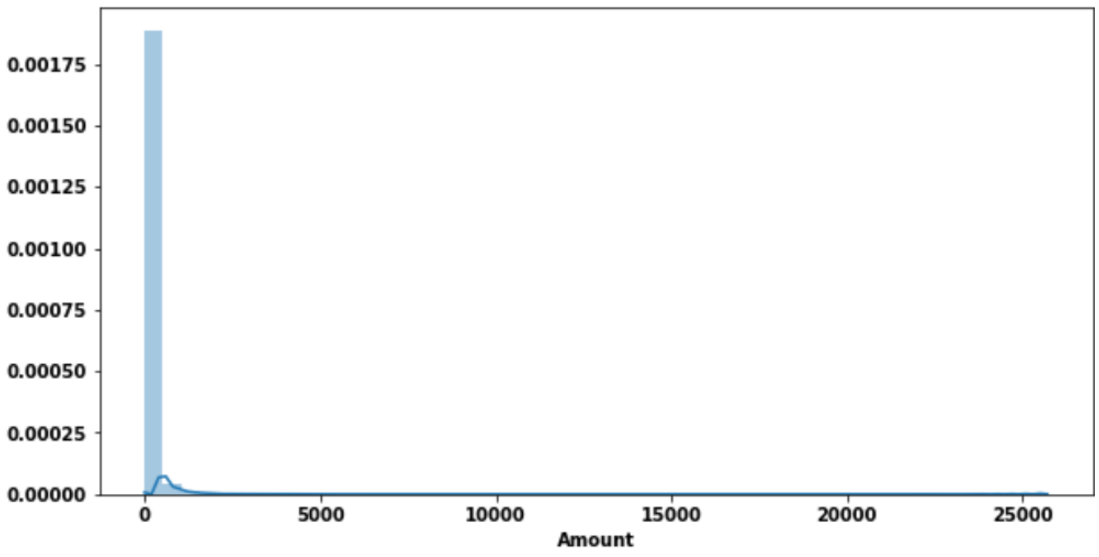


Figure 4: Distribution of the Monetary Value feature - Histogram

The data set contains 284,807 transactions. The mean value of all transactions is $88.35 while the largest transaction recorded in this data set amounts to $25691.16. However, as can be seen, based on the mean and maximum, the distribution of the monetary value of all transactions is heavily right-skewed. The vast majority of transactions are relatively small and only a tiny fraction of transactions is close to the maximum

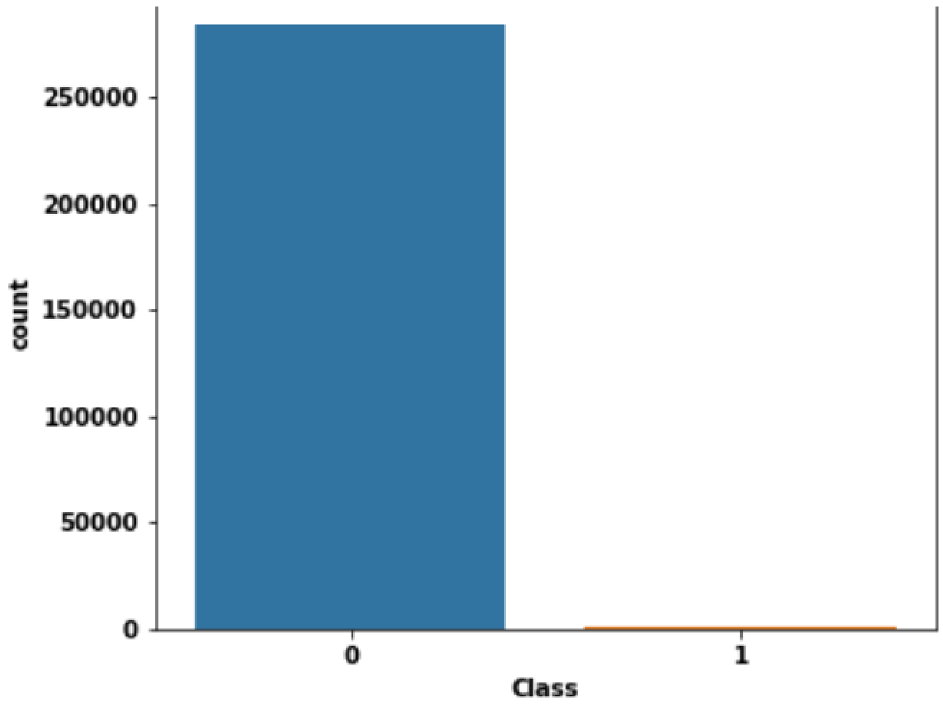


Figure 6: Class Bar Plot

As represented by the graph, the majority of the transactions are non-fraudulent. In fact, 98.48% of the transactions in this data set were not fraudulent while only 0.17% were fraudulent. The following visualization underlines this significant contrast.

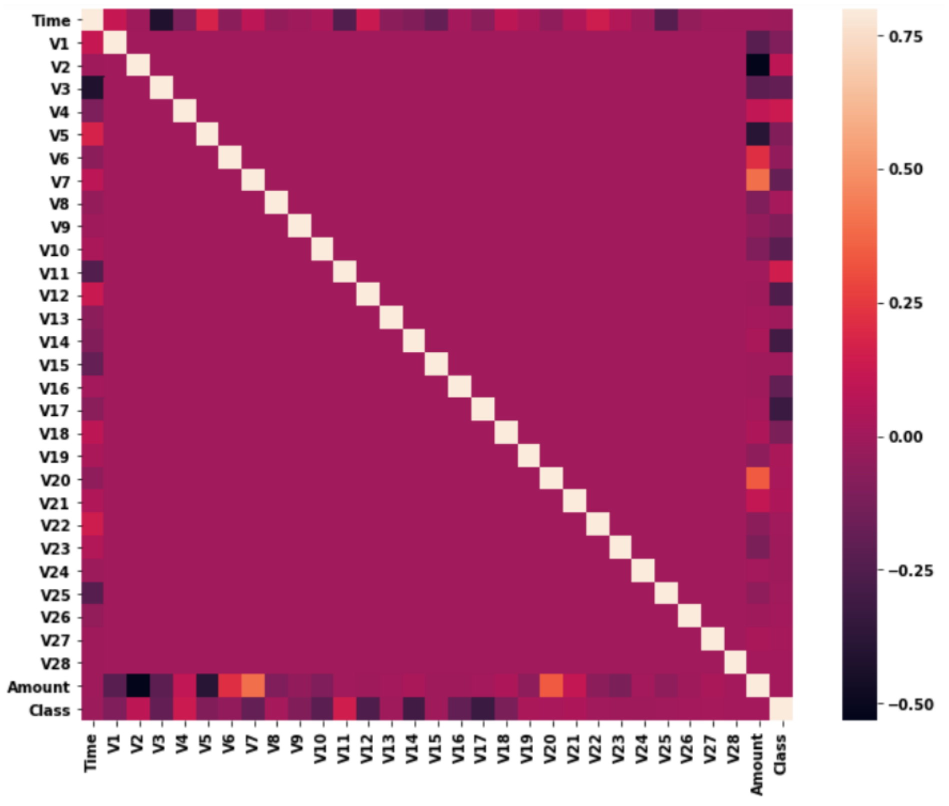


Figure 6: Heatmap

The correlation matrix helps graphically determine how features correlate with each other and help determine the best variables for the prediction. This Heatmap was used to make sure there wasn't any strong collinearity going on in the data. From the Heatmap above, few of the variables that showed high correlation were:

- Time and V3 (-0.42)
- Amount and V2 (-0.53)
- Amount and V4 (0.4)

While these correlations are relatively high in comparison to the others, this is not high enough to run the risk of multicollinearity.

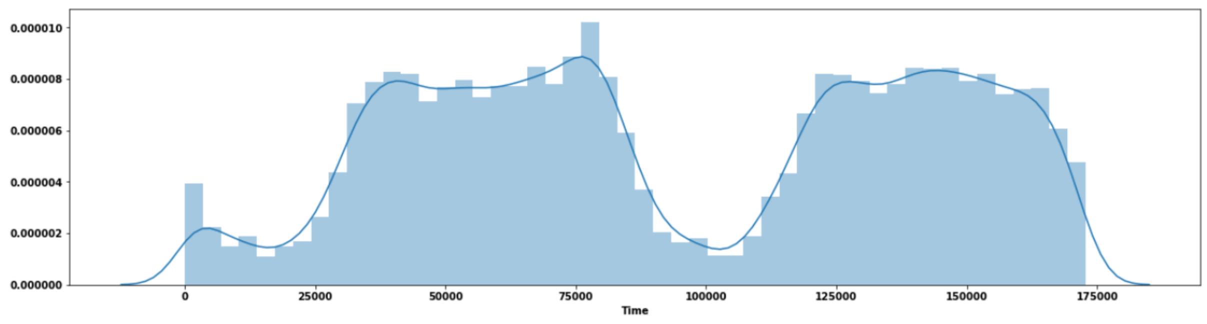


Figure 7: Time Histogram

The time is recorded in the number of seconds since the first transaction in the data set. Therefore, it can be concluded that this data set includes all transactions recorded over the course of two days. As opposed to the distribution of the monetary value of the transactions, it is bimodal. This indicates that approximately 28 hours after the first transaction there was a significant drop in the volume of transactions. While the time of the first transaction is not provided, it would be reasonable to assume that the drop in volume occurred during the night.

3.2.3 Verifying Data Quality

Essentially, the biggest problem in dealing with Credit card fraud detection scientifically is that real data is hardly ever available for exploration, due to the issue of confidentiality (M. Zareapoor, Application of credit card fraud detection: Based on bagging ensemble classifier, 2015). Nonetheless, the dataset consists of numerical values for the 28 Principal Component Analysis (PCA) transformed features, namely V1 to V28. Hence allowing analyses to be done on the data. However, this causes a lack of metadata about the original features being provided hence pre-analysis or feature study could not be done.

There are no missing values in the dataset, however, is highly imbalanced and skewed. It consists of 492 frauds in a total of 284,807 observations. This resulted in only 0.172% of fraud cases. This skewed set is justified by the low number of fraudulent transactions.

Lastly, the only coding inconsistencies that were faced were for 'Time and 'Amount' features are they were not transformed data.

3.3 Data Preparation

In the dataset used, before it was made public, due to data privacy issues, features were already transformed using PCA (Principal Component Analysis (A. Hervé, 2010) for dimensionality reduction. Because of the unknown meaning of the original or the constructed features, there was not enough information for the creation of additional features. The only other pre-processing done on the dataset was scaling the feature Amount. It was standardized by removing mean and scaled to unit variance.

3.4 Modeling

Before starting with the Modelling, the 'Time' feature did not indicate the actual time of the transactions but rather is more of a list of data in chronological order, it represented the difference between the transactions. Hence the assumption that 'Time' feature has little or no significance in classifying a fraud transaction. Consequently, the 'Time' column was removed from further analysis.

Furthermore, Fraudulent behavior tends to alter over time in order to avoid detection. Therefore, credit card fraud detection predictive model should not be static, i.e. constructed once and never updated (M. Zareapoor, Application of credit card fraud detection: Based on bagging ensemble classifier, 2015)

To start off the modeling, the dataset was first split into training and test data. The training data was set to 70% and test data to 30%.

The following models were implemented to train the dataset so it could predict future fraudulent transactions:

3.4.1 Random Forest:

The random forest consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

The Random Forest Classifier class of the sklearn.ensemble library was used to solve the classification problem.

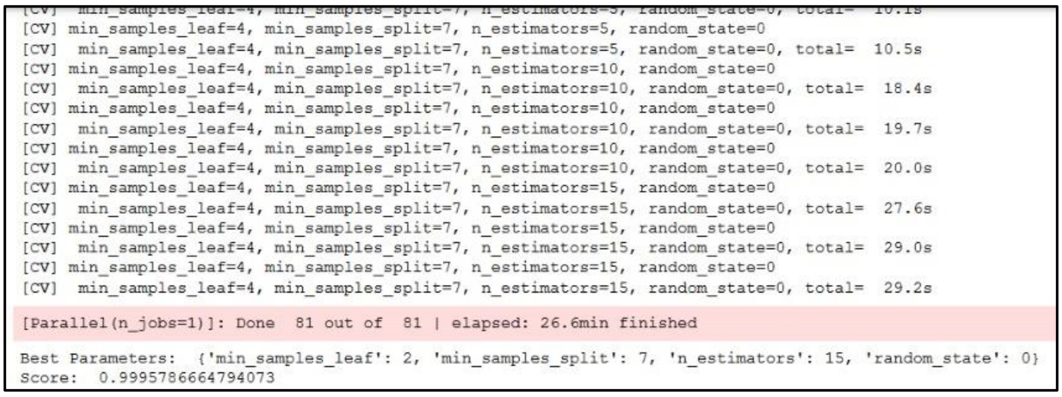
3.4.1.1 GridSearch Cross-Validation:

A GridSearchCV with 3 folds was used to pick the optimal parameters from the grid search and used it with the estimator i.e. Random Forest. The parameters used for this were:

- n_estimators specifying the number of trees in the random forest which was set to [5,10,15]
- min_samples_split specifying the minimum number of samples required to split a node which was set to [2, 5, 7]
- min_samples_leaf specifying the minimum number of samples required at each leaf node which was set to [1, 2, 4]

params = {'n_estimators': [5,10,15], 'min_samples_split': [2,5,7],'min_samples_leaf': [1,2,4], 'random_state':[0]}

rfc_grid = GridSearchCV (rfc, params, cv = 3, verbose = 2)



```
[CV]  min_samples_leaf=4, min_samples_split=7, n_estimators=5, random_state=0, total=  10.1s
[CV] min_samples_leaf=4, min_samples_split=7, n_estimators=5, random_state=0
[CV]  min_samples_leaf=4, min_samples_split=7, n_estimators=5, random_state=0, total=  10.5s
[CV] min_samples_leaf=4, min_samples_split=7, n_estimators=10, random_state=0
[CV]  min_samples_leaf=4, min_samples_split=7, n_estimators=10, random_state=0, total=  18.4s
[CV] min_samples_leaf=4, min_samples_split=7, n_estimators=10, random_state=0
[CV]  min_samples_leaf=4, min_samples_split=7, n_estimators=10, random_state=0, total=  19.7s
[CV] min_samples_leaf=4, min_samples_split=7, n_estimators=10, random_state=0
[CV]  min_samples_leaf=4, min_samples_split=7, n_estimators=10, random_state=0, total=  20.0s
[CV] min_samples_leaf=4, min_samples_split=7, n_estimators=15, random_state=0
[CV]  min_samples_leaf=4, min_samples_split=7, n_estimators=15, random_state=0, total=  27.6s
[CV] min_samples_leaf=4, min_samples_split=7, n_estimators=15, random_state=0
[CV]  min_samples_leaf=4, min_samples_split=7, n_estimators=15, random_state=0, total=  29.0s
[CV] min_samples_leaf=4, min_samples_split=7, n_estimators=15, random_state=0
[CV]  min_samples_leaf=4, min_samples_split=7, n_estimators=15, random_state=0, total=  29.2s

[Parallel(n_jobs=1)]: Done  81 out of  81 | elapsed: 26.6min finished

Best Parameters:  {'min_samples_leaf': 2, 'min_samples_split': 7, 'n_estimators': 15, 'random_state': 0}
Score:  0.9995786664794073
```

3.4.1.2 Data Training and Testing:

The data was then trained and tested on the model to get the best parameters.

rfc_fit = rfc_grid.fit(x_train, y_train)

score = rfc_grid.score(x_test, y_test)

The following best parameters and score were achieved:

Best Parameters: {'min_samples_leaf': 2, 'min_samples_split': 7, 'n_estimators': 15, 'random_state': 0}

Score: 0.9995786664794073

The model was then trained again on the new best parameters achieved above:

rfc2 = RandomForestClassifier(min_samples_leaf = 2, min_samples_split = 7, n_estimators = 15,random_state=0)

rfc2.fit(x_train,y_train)

y_pred_rfc = rfc2.predict(x_test)

3.4.1.3 Evaluation:

Furthermore, the five metrics used to evaluate the performance of the algorithm were accuracy score, precision score, recall score, f1 score and confusion matrix. The final results were:

Accuracy: 0.9995786664794073

Precision: 0.9310344827586207

Recall: 0.7941176470588235

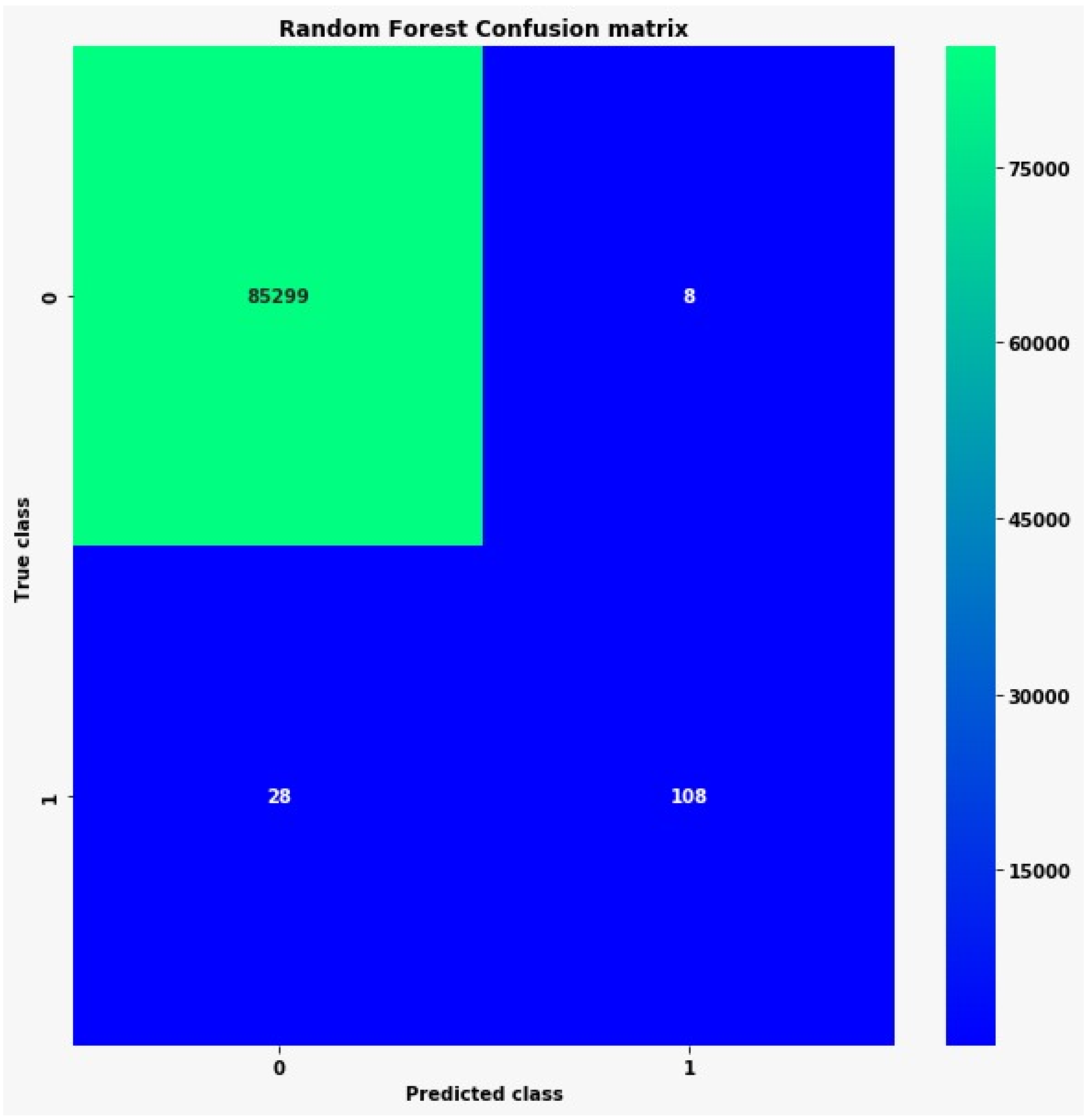
F1 Score: 0.8571428571428571

RMSE: 0.020526410319212558

3.4.1.4

3.4.1.5 Confusion Matrix:

The following confusion matrix was generated by this model:



3.4.2 Decision Tree:

A decision tree is a flowchart-like tree structure where an internal node represents a feature, the branch represents a decision rule, and each leaf node represents the outcome.

The Decision Tree Classifier class of the sklearn.tree library and GridSearchCV class of the sklearn. model_selection library were used in this algorithm.

3.4.2.1 Decision Tree Classifier:

The Decision Tree Classifier class is capable of performing multi-class classification on a dataset.

The parameters used for this class were max_depth and min_samples_leaf which controls the size of the tree.

- The max_depth parameter tells how long the tree would be. which was set to [2,4,10]
- min_samples_split parameter tells the minimum no. of samples an internal node is supposed to have to further split into nodes which were set to [2,3,4,6,8,10].

This means the algorithm used all these values turn by turn. If these values are set to default, it would lead to fully grown and unpruned trees.

```
params = {'max_depth': [2,4,10], 'min_samples_split':[2,3,4,6,8,10]}
```

3.4.2.2 Data Training and Testing:

Furthermore, Grid Search with 3Fold Cross validation was used to train and test the estimator i.e. Decision Tree to get the optimal combination of best parameters. The score function of GridSearchCV class was used to measure the performance of the algorithm and best_params_ was used to get the best parameters.

```
clf_dtc = GridSearchCV(dtc, params, cv = 3)

dtc_fit= clf_dtc.fit(x_train, y_train)

score = clf_dtc.score(x_test, y_test)
```

The values for the best parameters and the score were:

```
Best Parameters:  {'max_depth': 4, 'min_samples_split': 2}

Score:  0.9994499256814484
```

After getting the above results, the model was trained again on the best parameters.

```
dtc2 = DecisionTreeClassifier(max_depth= 4, min_samples_split= 2, random_state=0)

dtc2.fit(x_train, y_train)

dtc2_score = dtc2.score(x_test, y_test)

y_pred_dtc = dtc2.predict(x_test)
```

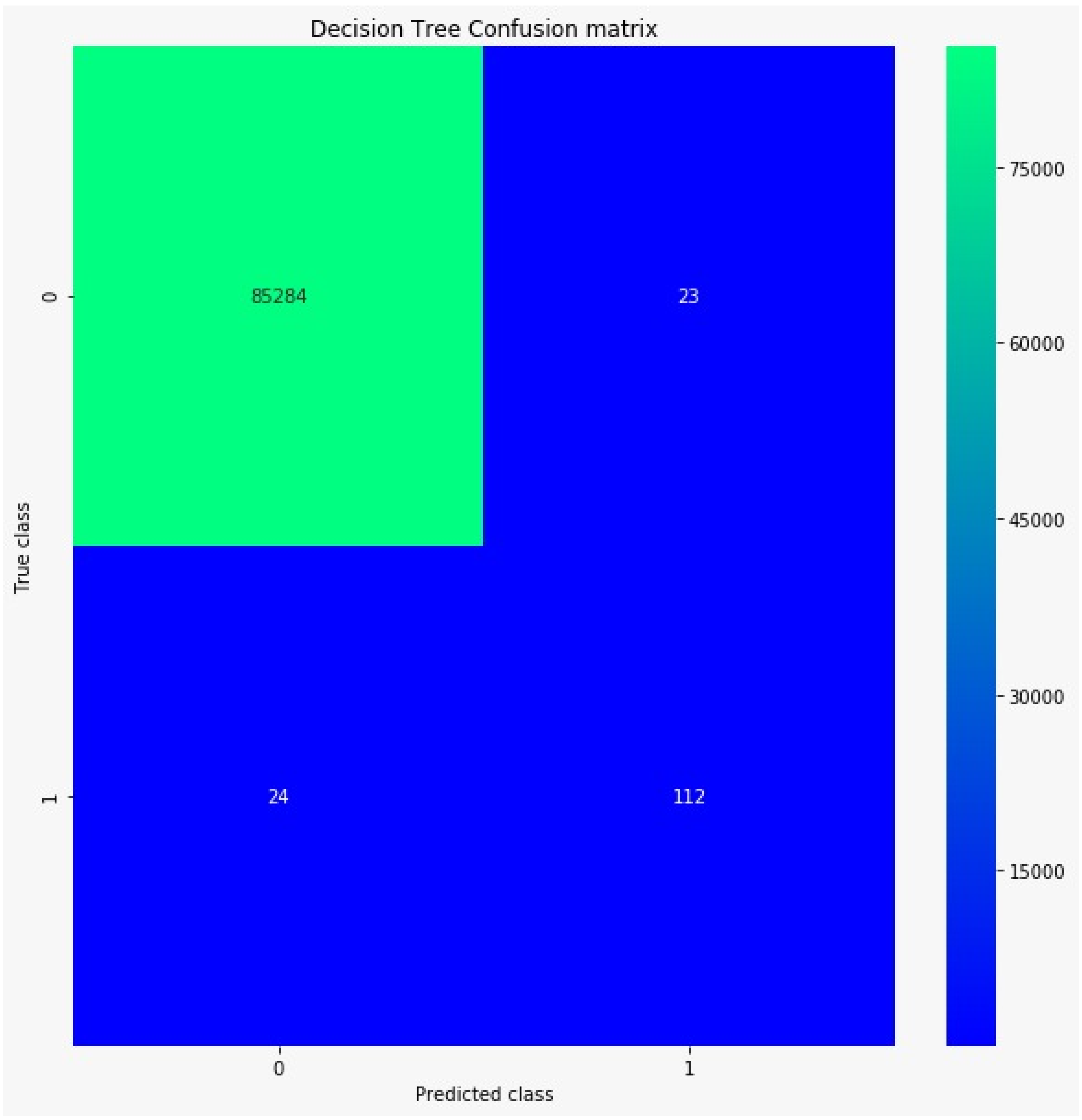
3.4.2.3 Evaluation:

The metrics used to evaluate the performance of the algorithm were accuracy score, precision score, recall score, F1 score, RMSE and confusion matrix. The final results were:

```
Score:  0.9994499256814484

Precision:  0.8296296296296296

Recall:  0.8235294117647058

F1 Score:  0.8265682656826567

RMSE:  0.023453663222438175


Proba: array ([[9.99728426e-01, 2.71573770e-04],

    [9.99728426e-01, 2.71573770e-04],

    [9.99728426e-01, 2.71573770e-04],

    ...,

    [9.99728426e-01, 2.71573770e-04],

    [9.99728426e-01, 2.71573770e-04],

    [9.99728426e-01, 2.71573770e-04]])
```

3.4.2.4 Confusion Matrix:

The following confusion matrix was generated by this model:


Decision Tree Confusion matrix
85284
23
24
112
True class
Predicted class
0
1
0
1
75000
60000
45000
30000
15000

3.4.3 Gradient Boosting:

Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model.

The Gradient Boosting Classifier class of the sklearn.ensemble library was used to solve the classification problem.

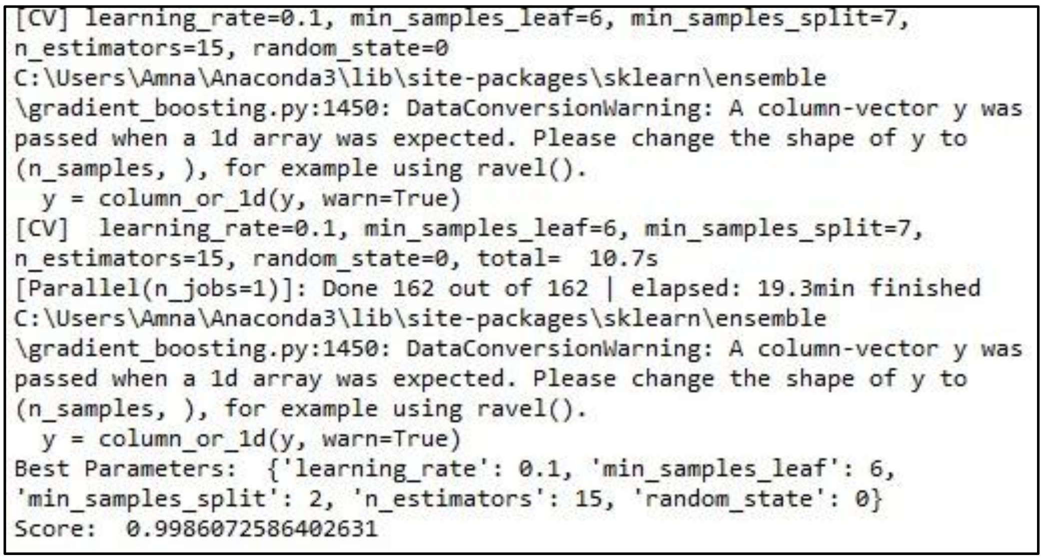
3.4.3.1 GridSearch Cross-Validation:

A GridSearchCV with 3 folds was used to pick the optimal parameters from the grid search and used it with the estimator i.e. Gradient Boosting. The parameters used for this were:

- n_estimators specifying the number of trees in the random forest which was set to [5,10,15]
- min_samples_split specifying the minimum number of samples required to split a node which was set to [2, 5, 7]
- min_samples_leaf specifying the minimum number of samples required at each leaf node which was set to [2, 4, 6]
- learning_rate which was set to [0.01, 0.1]
- random_state which was set to 0

```
params = {'n_estimators': [5,10,15], 'min_samples_split': [2,5,7], 'min_samples_leaf': [2,4,6], 'learning_rate': [0.01, 0.1],'random_state':0}
```

```
gdc_grid = GridSearchCV (gdc, params, cv = 3, verbose = 2)
```



```
[CV] learning_rate=0.1, min_samples_leaf=6, min_samples_split=7,
n_estimators=15, random_state=0
C:\Users\Amna\Anaconda3\lib\site-packages\sklearn\ensemble
\gradient_boosting.py:1450: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
[CV]  learning_rate=0.1, min_samples_leaf=6, min_samples_split=7,
n_estimators=15, random_state=0, total=  10.7s
[Parallel(n_jobs=1)]: Done 162 out of 162 | elapsed: 19.3min finished
C:\Users\Amna\Anaconda3\lib\site-packages\sklearn\ensemble
\gradient_boosting.py:1450: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
Best Parameters:  {'learning_rate': 0.1, 'min_samples_leaf': 6,
'min_samples_split': 2, 'n_estimators': 15, 'random_state': 0}
Score:  0.9986072586402631
```

3.4.3.2 Data Training and Testing:

The data was then trained and tested on the model to get the best parameters.

```
gdc_fit = gdc_grid.fit(x_train, y_train)
```

```
score = gdc_grid.score(x_test, y_test)
```

The following best parameters and score were achieved:

```
Best Parameters: {‘learning_rate’:0.1, 'min_samples_leaf':6,'min_samples_split':
2,'n_estimators': 15,’random_state’:0}

Score: 0.9986072586402631
```

The model was then trained again on the new best parameters achieved above:

```
gdc2 = GradientBoostingClassifier(learning_rate = 0.1, min_samples_leaf = 6,
min_samples_split = 2, n_estimators =15, random_state=0)

gdc2.fit(x_train, y_train)

gdc_score = gdc2.score(x_test, y_test)

y_pred_gdc = gdc2.predict(x_test)
```

3.4.3.3 Evaluation:

Furthermore, the metrics used to evaluate the performance of the algorithm were accuracy score, precision score, recall score, f1 score, RMSE and confusion matrix. The final results were:

Score: 0.9986072586402631

Precision: 0.8695652173913043

Recall: 0.14705882352941177

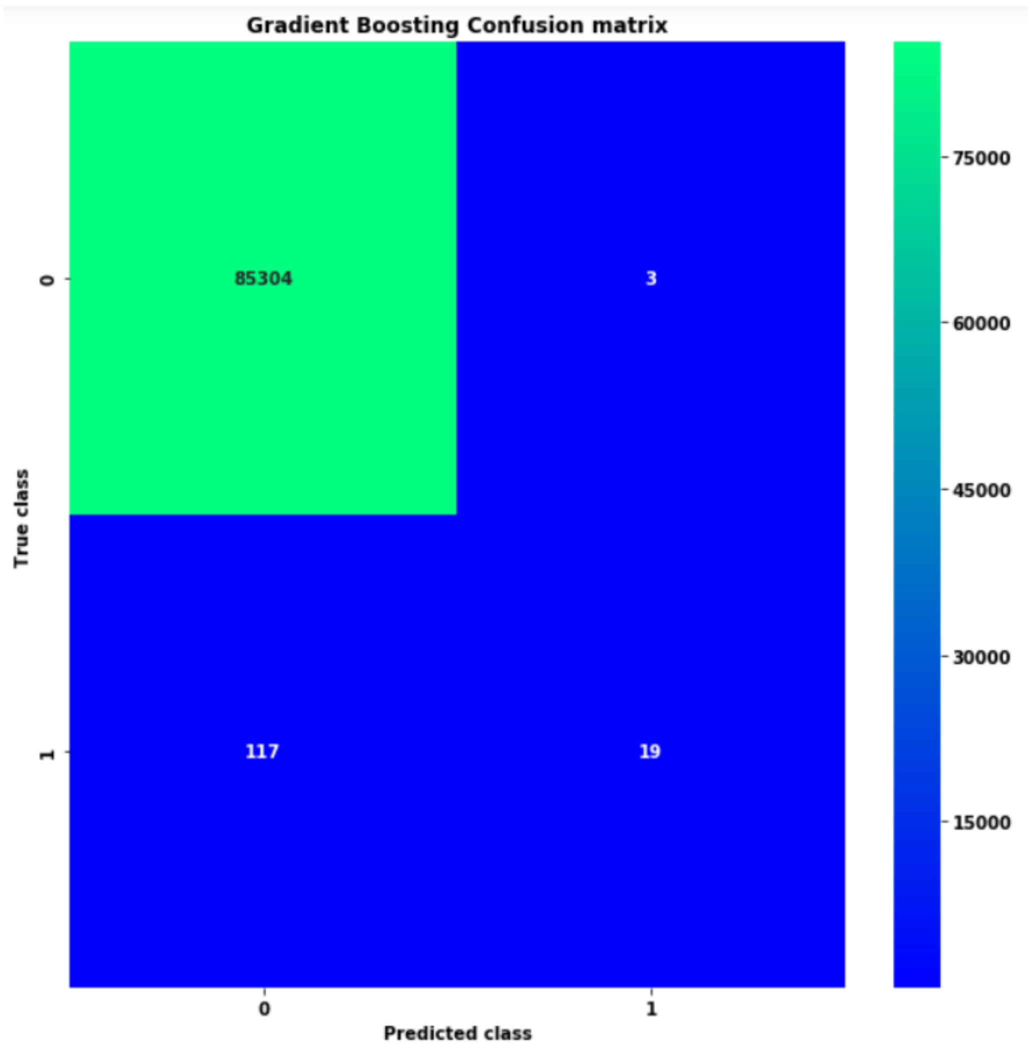
F1 score : 0.25157232704402516

RMSE: 0.037319450153196265

3.4.3.4

3.4.3.5 Confusion Matrix:

The following confusion matrix was generated by this model:



3.4.4 Neural Networks:

For the neural networks, Keras library was used. It is a simple tool for constructing a neural network.

3.4.4.1 Classes used:

The classes used from Keras library were Sequential which was imported from Keras. models library and Dense class which was imported from Keras.layers library. The sequential class specifies to Keras that a sequential model is being created and the output of each layer added is input to the next layer while the Dense class specifies that the model is a fully connected layer.

3.4.4.2 Layers:

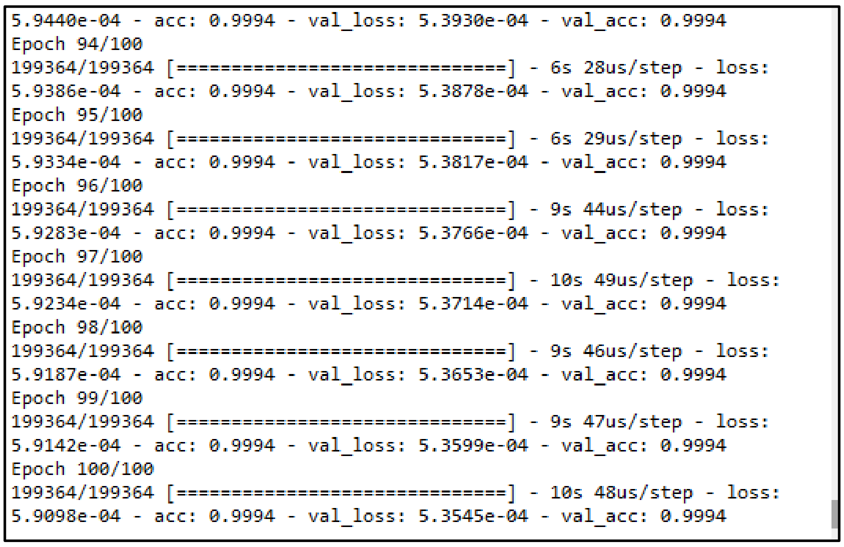
```
classifier = Sequential()
classifier.add(Dense(units = 15 , kernel_initializer = 'uniform', activation = 'relu', input_dim =
29))
classifier.add(Dense(units = 15, kernel_initializer = 'uniform', activation = 'relu'))
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
```

From the dense class, model.add function was used to add layers to the neural network. The arguments used for the first layer were:

- output dimension which was 15 specifying the number of neurons
- input dimension which was 30 specifying the number of input values and the activation function which was relu which gives results from 0 to infinity. It is the most widely used activation function

The second layer is similar, the input dimension does not need to be specified as it was already being defined that the model is sequential so Keras will automatically consider the input dimension to be the same as the output of the last layer i.e. 15.

In the output layer, the output dimension is 2 specifying the number of classes and the activation function used was sigmoid. Sigmoid function gives values between 0 and 1 and is appropriate for classification problems.



```
5.9440e-04 - acc: 0.9994 - val_loss: 5.3930e-04 - val_acc: 0.9994
Epoch 94/100
199364/199364 [==============================] - 6s 28us/step - loss:
5.9386e-04 - acc: 0.9994 - val_loss: 5.3878e-04 - val_acc: 0.9994
Epoch 95/100
199364/199364 [==============================] - 6s 29us/step - loss:
5.9334e-04 - acc: 0.9994 - val_loss: 5.3817e-04 - val_acc: 0.9994
Epoch 96/100
199364/199364 [==============================] - 9s 44us/step - loss:
5.9283e-04 - acc: 0.9994 - val_loss: 5.3766e-04 - val_acc: 0.9994
Epoch 97/100
199364/199364 [==============================] - 10s 49us/step - loss:
5.9234e-04 - acc: 0.9994 - val_loss: 5.3714e-04 - val_acc: 0.9994
Epoch 98/100
199364/199364 [==============================] - 9s 46us/step - loss:
5.9187e-04 - acc: 0.9994 - val_loss: 5.3653e-04 - val_acc: 0.9994
Epoch 99/100
199364/199364 [==============================] - 9s 47us/step - loss:
5.9142e-04 - acc: 0.9994 - val_loss: 5.3599e-04 - val_acc: 0.9994
Epoch 100/100
199364/199364 [==============================] - 10s 48us/step - loss:
5.9098e-04 - acc: 0.9994 - val_loss: 5.3545e-04 - val_acc: 0.9994
```

3.4.4.3 *Data Compiling*

The compile function is used for configuring the model for training.

```
classifier.compile(optimizer = 'SGD', loss = 'mean_squared_error', metrics = ['accuracy'])
```

The parameters used here were:

- Optimizer
- Loss which is the objective that the model tried to minimize which was set to mean squared error
- Metrics is used to specify the way the performance of the neural network is to be judged which was set to accuracy

3.4.4.4 *Data Training:*

After creating the neural network, the model was trained for a fixed number of epochs (iterations on a dataset) using the fit function.

```
classifier.fit(x_train, y_train, batch_size = 32, epochs = 100, validation_data=(x_test, y_test))
```

Here the parameters used were:

- Input data which was set to x_train
- Target data to be trained which was set to y_train
- Number of epochs i.e. the number of iterations over the entire x and y data provided to train the model which was set to 100
- Batch size which was set to 32. Usually, the datasets are very big and the model cannot fit complete data at once, so batch size is used. This divides the data into batches each of size equal to batch_size. Now only this number of samples get loaded into memory and processed. Once one batch if finished it is flushed from memory and the next batch gets processed
- Validation data i.e. the data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. The validation data was set to x_test, y_test

3.4.4.5 Data Testing and Evaluation:

Finally, the model performance was tested on the test data

```
y_pred_nn = classifier.predict(x_test).ravel()

proba_nn = classifier.predict_proba(x_test).ravel()

score = classifier.evaluate(x_test, y_test)
```

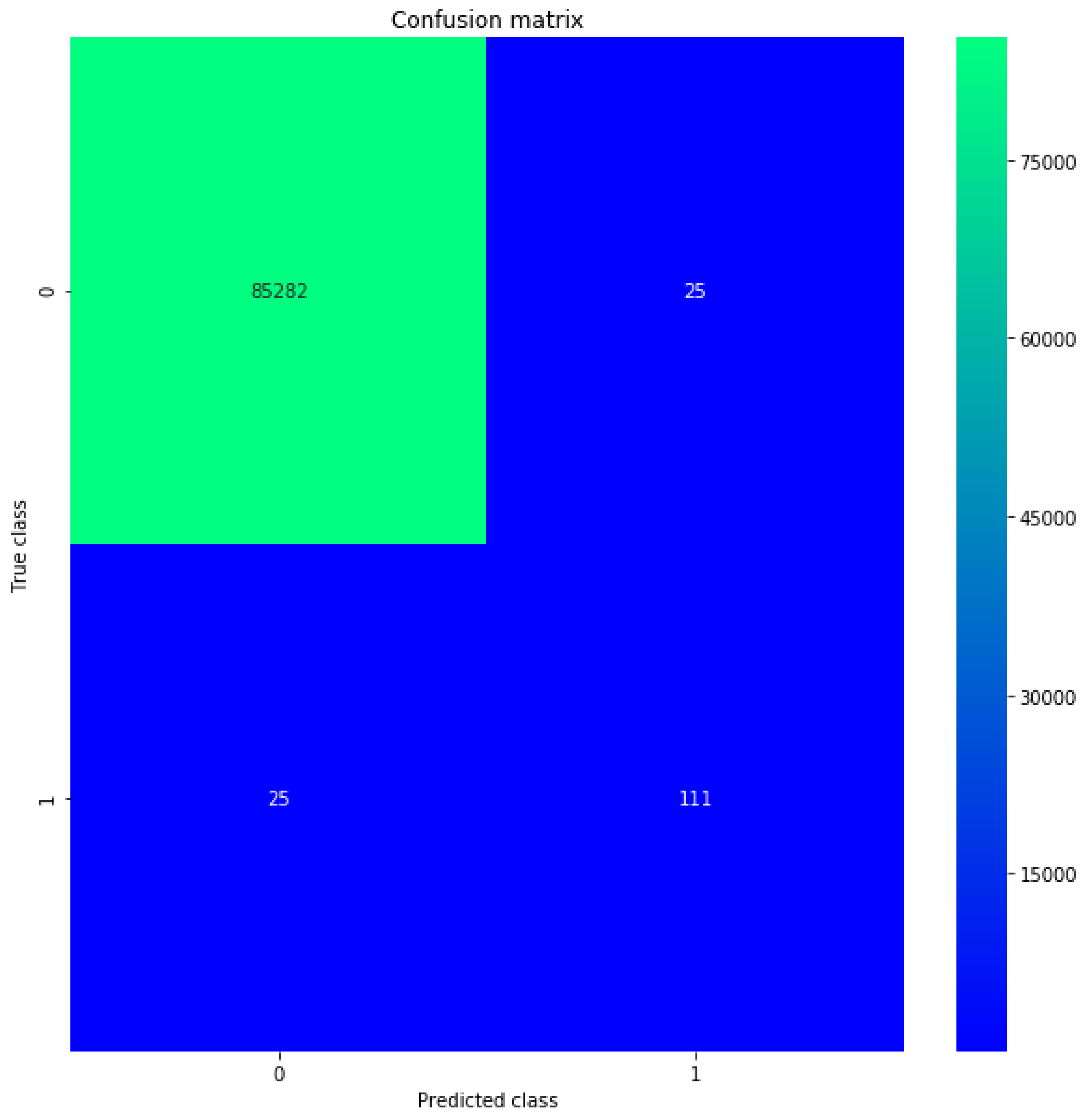
3.4.4.6 Evaluation:

The following accuracy score was achieved by this model:

Accuracy: 0.999414814554732

3.4.4.7 Confusion Matrix:

The following confusion matrix was generated by this model:



3.5 Evaluation

As mentioned before the accuracy is not a suitable metric for credit card fraud detection, due to class imbalance. Metrics that are commonly used for fraud detection combine precision, recall (true positive rate) and fall-out (false positive rate). Metrics are based on a well-known machine learning concept, confusion metrics with the following four categories defined:

TP = True Positive. Fraudulent transactions the model predicts as fraudulent.

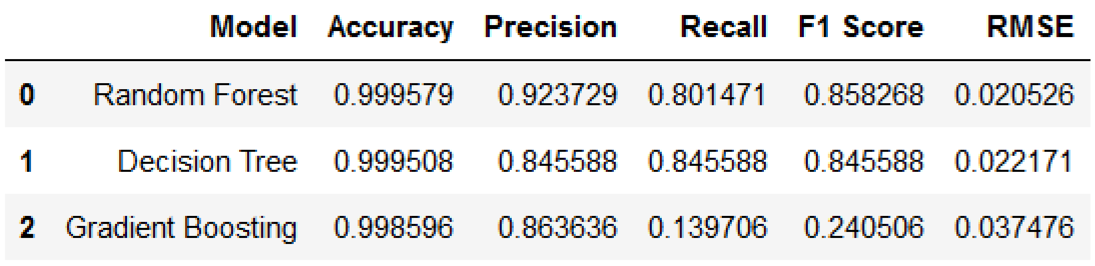
TN = True Negative. Non-fraud transactions the model predicts as non-fraud.

FP = False Positive. Non-fraud transactions the model predicts as fraudulent.

FN = False Negative. Fraudulent transactions the model predicts as non-fraudulent.

Following metrics were used to measure the performance of the models implemented:

Furthermore, Accuracy, 'Precision, Recall, F1 Score and RMSE was used for models.



	Model	Accuracy	Precision	Recall	F1 Score	RMSE
0	Random Forest	0.999579	0.923729	0.801471	0.858268	0.020526
1	Decision Tree	0.999508	0.845588	0.845588	0.845588	0.022171
2	Gradient Boosting	0.998596	0.863636	0.139706	0.240506	0.037476

The main goal here is to improve the classification of false negatives – recall without hurting the classification of false positives – precision. The reason why we need to improve evaluation of both types (FP, FN) of the incorrectly classified examples is that each type causes different cost, making this cost-sensitive evaluation. The cost of false negatives is of financial kind; its significance varies depending on the amount of the transaction. On the other hand, cost of false positives is measured in terms of customer dissatisfaction. Metric that combines TPR and FPR is called The Receiver Operating Characteristic (ROC) curve and is represented by calculating area under the ROC curve (AUC). AUC is a well-accepted measure for handling class imbalance

3.6 Deployment

Lastly, since the data is highly imbalanced, methods such as oversampling and under sampling need to be considered; they do alter the size of the dataset used for training. However, while the level of imbalance is reduced, problems of overfitting and ignoring useful data appear, respectively (X.-Y. Liu, 2009). A possible improved approach that can be used is a complex sampling method SMOTE, which oversamples the minority class generating synthetics examples by interpolating k minority class nearest neighbours

Through this process, the classifier builds larger decision regions that contain nearby examples from the minority class, which would possibly show improvements in the application.

Furthermore, since the static training approach is not a long-term solution, the next step would be to investigate incremental learning on a realistic dataset. Richer datasets are not publicly available hence to further the research, the best alternative would be to work on producing realistic synthetic data.

4 Conclusion

In this report, the main issues were outlined in the credit card fraud detection field and state-of-the-art solutions were proposed. Using the only publicly available dataset suitable for the credit card fraud detection, the machine learning algorithms implemented and used for measuring performances were: Random Forest, Decision Tree, Gradient Boosting, and Neural Networks. The selected algorithms belong to the most often used machine learning algorithms in credit card fraud detection. The following measures were used to evaluate the performance of the algorithms: area under the ROC curve (AUC), precision and recall score, F1 score and confusion matrix. Form the results presented in the report, it is evident that while all of the models showed relatively positive and great results with a high percentage of accuracy and score, Neural Network provided the best and most accurate results due to its usage of ROC curve to determine whether the cases were fraudulent or non-fraudulent. The complex nature of Neural Networks and ROC/ AUC was evidently the reason behind better results.

In essence although there are several fraud detection techniques available today but none is able to detect all frauds completely when they are actually happening, they usually detect it after the fraud has been committed. This happens because a very minuscule number of transactions from the total transactions are actually fraudulent in nature. So we need a technology that can detect the fraudulent transaction when it is taking place so that it can be stopped then and there and that too in a minimum cost. So the major task of today is to build an accurate, precise and fast detecting fraud detection system for credit card frauds that can detect not only frauds happening over the internet like phishing and site cloning but also tampering with the credit card itself i.e. it signals an alarm when the tampered credit card is being used. Some techniques like Artificial Neural Network have high detection rates and gives high accuracy they are very expensive to train Some techniques like decision tree and support vector gives better results on sampled and pre-processed data whereas some techniques like logistic regression and fuzzy systems give better accuracies with raw un-sampled data.

A solution to these gaps by creating a hybrid of various techniques that are already used in fraud detection to cancel out their limitations and get enhanced performance.

5 Bibliography

- A. Hervé, L. J. (2010). Principal Component Analysis. Wiley Interdiscip. Rev. Comput. Stat. 2, vol. 2.

- Capgemini. (2018). World Payment Report 2018. 56.

- Data Minning Concepts. (n.d.).

- M. Zareapoor, P. S. (2015). In *Application of credit card fraud detection: Based on bagging ensemble classifier* (pp. 679-686). Procedia Comput. Sci., vol. 48.

- M. Zareapoor, P. S. (2015). Application of credit card fraud detection: Based on bagging ensemble classifier. Procedia Comput. Sci., vol. 48.

- P. H. Tran, K. P. (2018). Real Time Data-Driven Approaches for Credit Card Fraud Detection. Int. Conf. E-bus. Appl.

- S. Bhattacharyya, S. J. (2011). Data mining for credit card fraud: A comparative study. Decis. Support Syst.

- T. P. Bhatla, V. Prabhu, A. Dua. (2003). Understanding Credit Card Frauds. Tata Consult.

- X.-Y. Liu, J. W.-H. (2009). Exploratory Undersampling for Class-Imbalance Learning. In *MAN Cybern. B Cybern., vol. 39* (pp. 539-550).