Feedforward neural network with hyperparameter optimisation

Dataset: Credit Card Fraud detection dataset.

Introduction

Hyperparameters are variables that determine the network topology (for example, the number of hidden units) and how the network is trained (Eg: Learning Rate). Prior to training, hyperparameters are set (before optimising the weights and bias). The hyperparameter settings of modern machine learning and data mining systems have a significant impact on their performance. When applying a new algorithm to a dataset, it's usually unclear which hyperparameters should be modified, what good ranges to use, and which values within those ranges are most likely to produce good results. At the moment, these decisions are usually made based on a combination of intuition and trial and error. While different post-hoc analysis techniques exist to discover what were the most relevant hyperparameters and which of their values likely to generate good performance for a specific dataset and algorithm.

This research project will demonstrate performance of different classifiers compared with the feed forward neural network using same dataset. Initially, the neural network would be trained for the dataset to predict the fraud transaction possibilities by analysing previous transactions occurred in 2013.Hyperparameters for the neural network are tuned based on trial and error which provides an idea on most suitable values for the parameters to create a model which produce good results for a particular dataset. Later, the trained neural network performance will be compared with other well-known classifiers such as Decision Tree, Naïve Bayes and Random Forest to identify which of their performance yields better accuracy for the given dataset.

The outcome of this project is to answer the following questions:

1. Which hyperparameters of the Feed Forward Neural network are most important for empirical performance of the model?
2. What values of these hyperparameters are most likely to produce good results?
3. Which classifiers are the most accurate on the dataset in order to achieve good results?

Related Work

This section would cover the related work on hyperparameter tuning importance and classifiers.

Daniel[1] shows in his research paper that importance of neural network hyperparameter optimization. A bad choice of hyperparameters can cause a network's weights to converge slowly or not at all during training, resulting in a lot of wasted processing resources. Because these design decisions have been of particular importance to data scientists since they began working on neural networks, there is a body of theory dedicated to finding solutions to the problem of how to effectively maximise hyperparameters. Grid Search and Random Search are two traditional approaches to hyperparameter optimization [2]. Both of these approaches, however, have inefficiencies that more current approaches, such as evolutionary hyperparameter optimization algorithms like Population Based Training[3]. Breiman [4] explained how random forests may be used to determine attribute relevance in his research paper: if deleting an attribute from the dataset results in a decline in performance, this is an indication that the attribute was essential. The entropy, information gain, and properties of the data sets all have an impact on the Decision Tree classifier performance [5]. In fact, the Naïve Bayes classifier is surprisingly successful, as its classification choice is often correct even if its probability estimations are incorrect [6].

Background

Maniraj [7] proposed Credit Card Fraud Detection using Machine Learning and Data Science, which is based on fraud detection prediction using the Local Outlier Factor algorithm and the Isolation Forest Algorithm. The data used for the prediction was taken from the Kaggle dataset. Both algorithms' precision and accuracy scores are determined by this. Isolation forest has a 99.76 percent accuracy rate, whereas Local Outlier Factor has a 99.67 percent accuracy rate. The accuracy scores for both algorithms [7] is nearly 99.70 percent

Kartik's [8] research work focuses on a credit card fraud detection system that employs of data sets that includes 51149 regular transactions and 3312 fraudulent transactions. Support Vector Machine, K-nearest neighbour, Logistic Regression, Error Back Propagation Algorithm, and Gradient Boosting are some of the models that were used. These models have accuracy values ranging from 90% to 96%. The Gradient Boosting method came out on top by 96.9%.

The same data set from Kaggle [7] is used in our study, but with alternate algorithms other than mentioned in [7][8] such as Decision Tree, Nave Bayes, and Random Forest, which are then

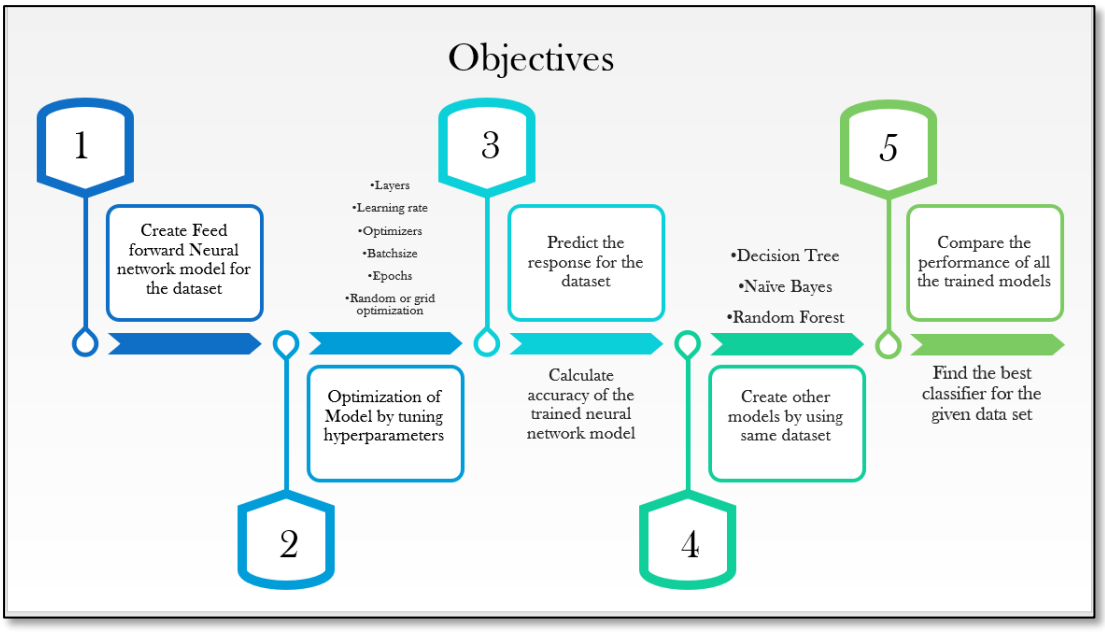
compared to the performance of Neural Network. Remember that accuracy is equal to the sum of True Negative and True Positive divided by the total dataset size. If 95% of the dataset is Negative (non-frauds), the network will intelligently predict that all will be Negative, resulting in 95% accuracy. Detecting Positive, on the other hand, is more important than detecting Negative when it comes to fraud detection. As a result, stronger metrics are required. This gives rise to the concept of hyperparameter significance. When a hyperparameter accounts for a significant portion of the variance, it is critical to set it correctly in order to get good performance, and it should be tuned effectively. When a hyperparameter isn't responsible for a large amount of variation, it's considered unimportant. This technique will be used as part of the proposed method.

As a result, this research project will examine which model is best for identifying and predicting fraudulent transactions with sufficient accuracy and precision while avoiding overfitting. It also looks at how to tune hyperparameters based on the dataset to improve accuracy and reduce loss.

Objectives

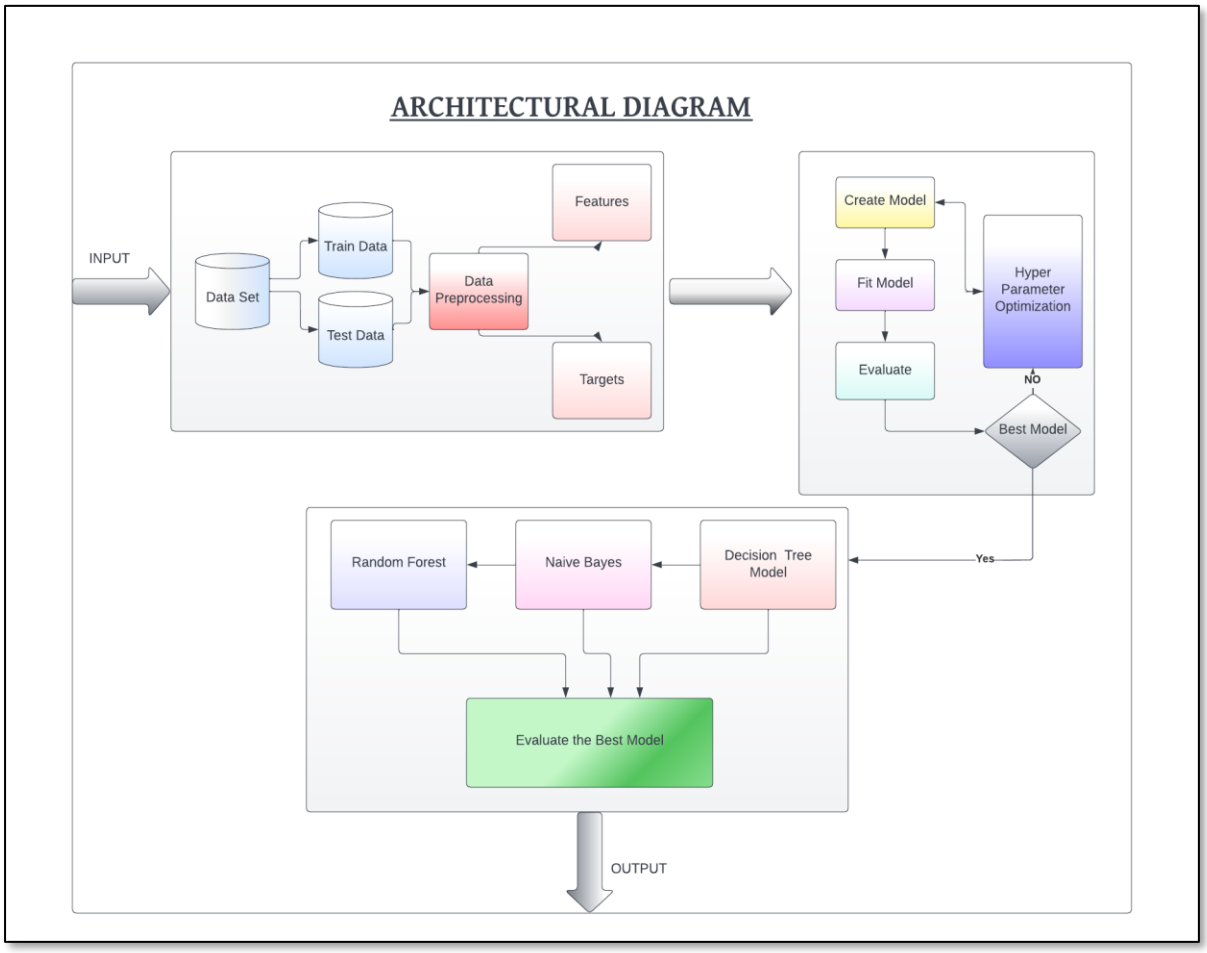
The objective of this research is to apply machine learning techniques to detect fraudulent credit card transactions over non-fraudulent transactions and to predict fraud fast and correctly. Machine Learning algorithms like as Decision Trees, Nave Bayes, Random Forests, and Neural Networks are used to detect fraudulent transactions in this project. Hyperparameter values are optimized to determine the most important hyperparameters for the neural network models. A comparison of various algorithms is carried out in order to find the best result.

- Create Feed forward Neural network model for the dataset



- Optimization of Model by tuning below hyperparameters
    - Layers
    - Learning rate
    - Optimizers
    - Batchsize
    - Epochs
    - Random or grid optimization
- Predict the response for the dataset
- Calculate accuracy of the trained neural network model
- Create other models by using same dataset
    - Decision Tree
    - Naïve Bayes
    - Random Forest
- Compare the performance of all the trained models
- Find the best classifier for the given data set

Methodology



The main goal of this project is to create the best algorithm for detecting outliers or frauds in credit card transactions which includes multiple machine learning and deep learning algorithms to the test, compare them, and pick the best one.

Implemented algorithms are:

- Neural Network
- Decision Tree
- Naïve Bayes
- Random Forest

We used a pre-existing dataset for this. The dataset includes data on transactions done by different cardholders. The dataset contains roughly 284,807 records, with just about 492 scammers among them. As a result, the dataset is highly imbalanced, with the positive class of fraud transactions accounting for only 0.172 percent of all transactions. Because of the PCA transformation, all of the feature's columns are numeric. As a result, their value varies from -1 to 1. As a result of PCA transformation, features columns V1, V2, V3,... V28 are obtained. The Feature Class column is a classification variable with values of 0 as Normal transaction and 1 as Fraud transaction.

The initial stage in the implementation of this research is to load the dataset. The data is then cleansed and normalised as part of the pre-processing procedure. The dataset is split into two sections: train data and test data, with the model being trained and tested in the middle. Finally, the system determines whether or not the transaction is valid.

After first level of evaluation, the model is more sensitive to detecting the majority class than the minority class. This proves the dataset is highly imbalanced. Under-sampling and over-sampling are the two most common approaches used to address class imbalance.

Dataset Summary

Data Set	Total Transactions	Normal Transactions	Fraud Transactions
Before Sampling	284807	284315	492
After Sampling	3000	2508	492

The technique used to address the class imbalance is **Under sampling approach.**

The most basic technique is to choose the majority class at random to balance out the minority class. However, data randomly removed from majority class may be beneficial in constructing a robust model. So, after sampling the new dataset consist of 2508 normal transactions and 492 fraudulent transactions. That provide a total 3000 records for the new dataset. Approximately 10 percent of original data set. This will construct a fresh dataset by randomly selecting the same number of non-frauds as the fraud. Since the data is now balanced, Hyperparameter optimization would be done on the newly created dataset to generate more accurate model with precision. Retrain the Neural Network model using the smaller data.

Pseudocode for Neural Network:

Step1: START
Step2: Load and Observe the dataset
    pd.read_csv(.csv) #read the dataset
    undersampling # needs to be done to rectify imbalanced data
    StandardScaler() #scaling & normalization of data
Step3: Data pre-processing
    train_test_Split() #spliting of data into test and train
Step 4: Create Model
    Model=Sequential() # define the model
Step5: Training the Model
    Dense() , Drop() # adding data to activation function in different layers
Step6: Compile the Model
    Model.compile() #adding hyperparameters like optimiser, epoch and batch size
Step7: Evaluate the model
    Model.evaluate() #Find the accuracy score and loss score
Step 8: STOP

Pseudocode for Decision Tree:

Step1: START
Step2: Read the dataset
    pd.read.csv(filename) #reads dataset file
Step3: Data cleaning & Data preprocessing
    Undersampling
    Scaled and normalized
Step4: Split the data
    train_test_Split() #spliting of data into test and train
Step5: Create Decision tree Model
    tree.decisiontreeclassifier()
Step6: Fit the model
    model.fit(train data) #fit train data
Step 7: Predict the Response
    model.predict(test data) #predict test data
Step 8: Fit Model
    model.fit(test data) #fit test data
Step 9: Predict the response
    model.predict(train data) #predict train data
Step10: Evaluate the model
    metrics.accuracy_score () #Find the accuracy score
Step 11: STOP

Pseudocode for Naïve Bayes:

Step1: START
Step2: Read the dataset
pd.read.csv(filename) #reads dataset file
Step3: Data cleaning & Data preprocessing
Undersampling
Scaled and normalized
Step4: Split the data
train_test_Split() #spliting of data into test and train
Step5: Initialize the Gaussian NB Model
Model= GaussianNB() #gaussian naïve bayes
Step6: Fit the Model
Model.fit(train data) #fit train data
Step7: Predict the Output
Model.predict(test data) #predict test data
Step8: Fit the Model
Model.fit(test data) #fit test data
Step9: Predict the Output
Model.predict(train data) #predict train data
Step10: Evaluate the model
metrics.accuracy_score () #Find the accuracy score
Step11: Display Classification report
Step12: STOP

Pseudocode for Random Forest:

Step1: START
Step2: Read the dataset
pd.read.csv(filename) #reads dataset file
Step3: Data cleaning & Data preprocessing
Undersampling
Scaled and normalized
Step4: Split the data
train_test_Split() #spliting of data into test and train
Step5: Initialize the Random Forest model
Model= RandomForestClassifier() # Random Forest
Step6: Fit the Model
Model.fit(train data) #fit train data
Step7: Predict the Output
Model.predict(test data) #predict test data
Step10: Evaluate the model
metrics.accuracy_score () #Find the accuracy score
Step12: STOP

Evaluation Measure:

The confusion matrix is used to calculate the precision and accuracy of the final output. In this system, there are two types of classes: actual and projected. These characteristics have an impact on the confusion metrics.

True Positive: When both values are positive, it is said to be true positive ie., 1.

True Negative: When both numbers are negative it is said to be true negative ie.,0.

False Positive: When true class is 0 and non-true class is 1, this is false positive.

False Negative: When the true class is 1 and the non-true class is 0, this is false negative.

•Precision measured as:

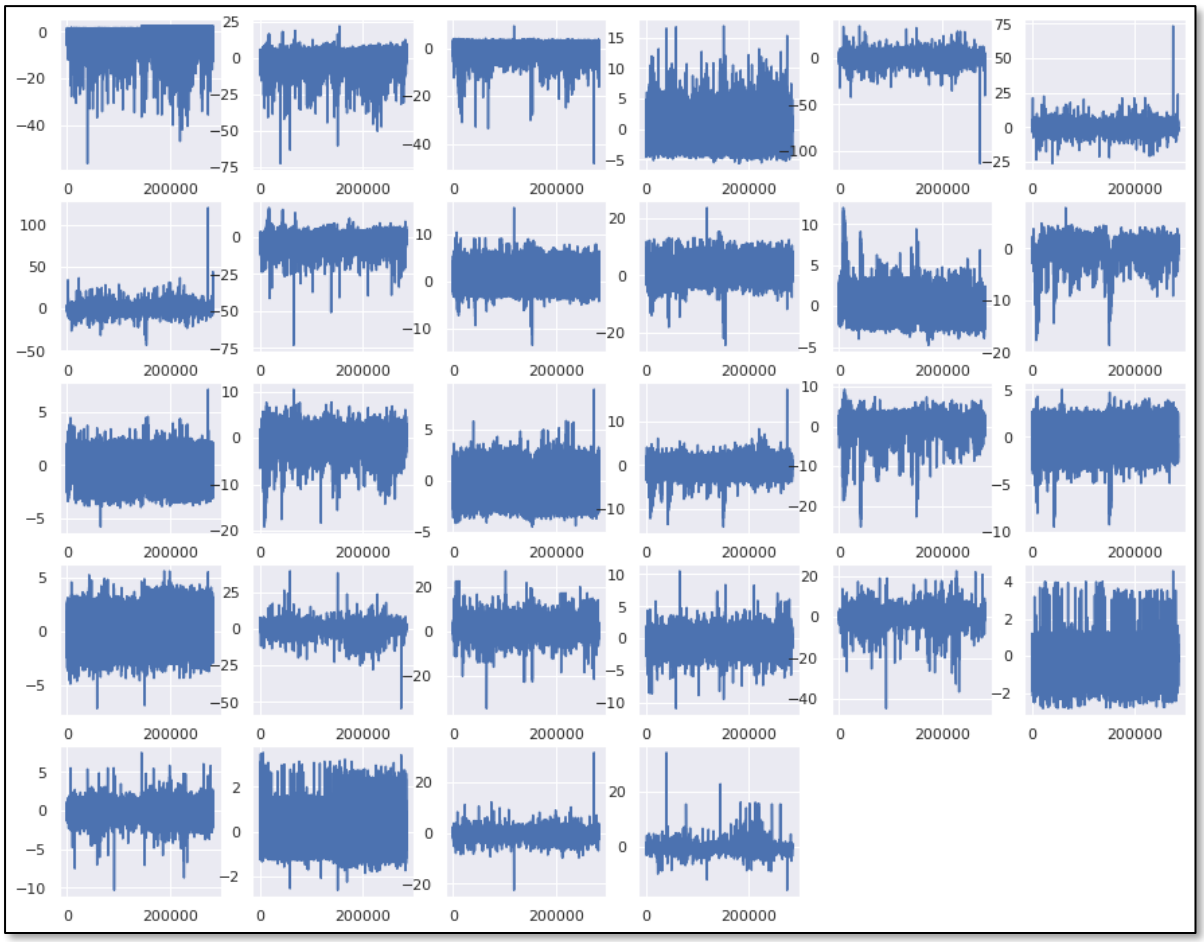
Precision = true positive / Actual result

Precision = true positive/(true positive + false positive)

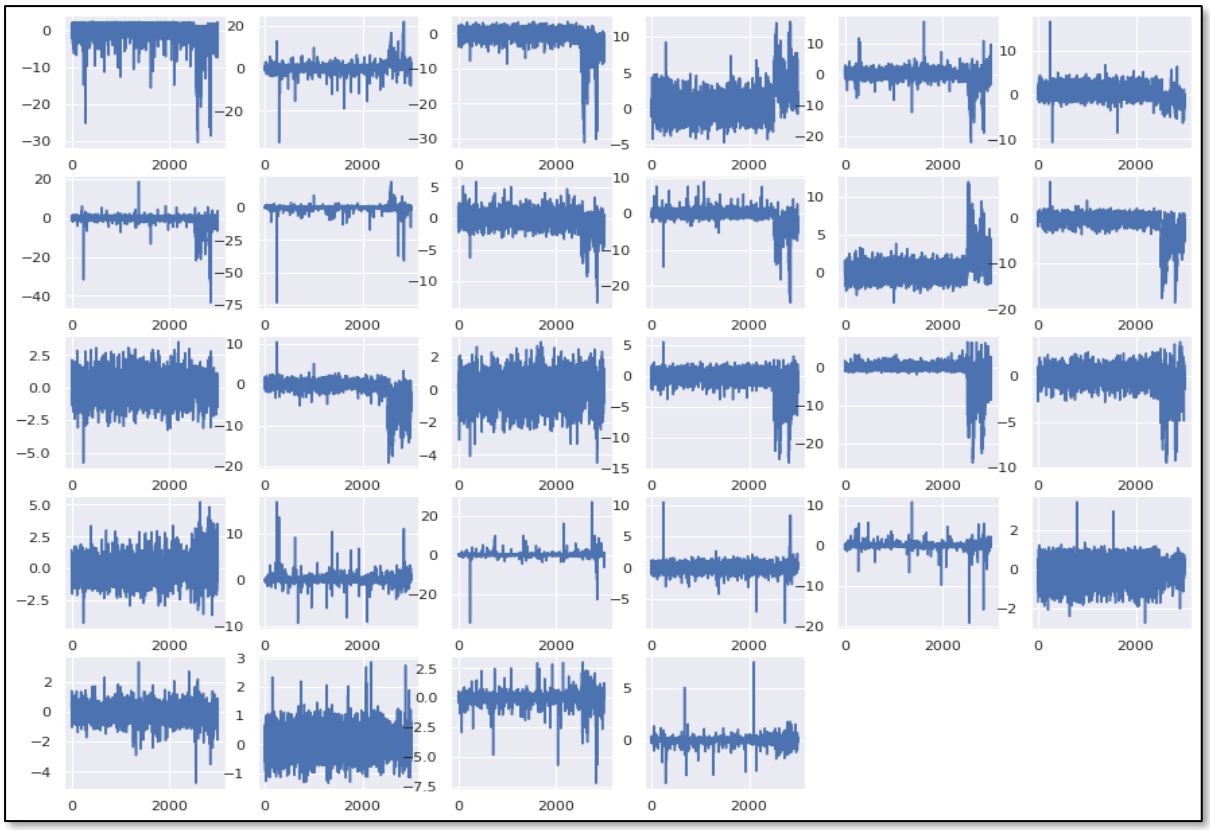
•Accuracy measured as:

Accuracy = (true positive + true negative)/ total

Before Sampling - Feature Plots (using subplots)



After Sampling - Feature Plots (using subplots)



Experiments

Dataset:

The dataset for the proposed system is available at www.kaggle.com. The data collection consisted of transactions made by customers of a European cardholders in 2013. There are 31 columns in all, 30 of which are characteristics and one of which is the target class, which decides whether or not the transaction is fraudulent.

Before Under-sampling:

```
Total Number of Normal Data: 284315
Total Number of Fraud Data :  492
```

Data Pre-processing

V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0.996326	-1.037305	0.046645	-1.019692	-0.298546	1.192329	-0.630693	0.386333	1.967784	...	-0.121127	-0.293312	-0.338080	-1.683625	0.618655	-0.623685	0.081146	0.025299	140.99	0
1.204209	0.175654	0.611244	0.556725	-0.560851	-0.795953	-0.067698	-0.046301	-0.161670	...	-0.197870	-0.630069	0.155305	0.518217	0.145421	0.066031	-0.035343	0.009825	0.99	0
-2.218371	0.845433	-1.280711	-0.585231	2.179790	0.067186	0.510032	0.904038	-1.421588	...	0.423592	1.177692	-1.084748	-1.504678	1.414456	0.444529	0.042937	-0.331718	5.92	0
-0.845259	1.185329	0.779094	-0.989750	0.668186	-0.006754	0.460673	-0.574037	-0.260058	...	0.487310	-1.111835	-0.047021	-0.947755	-0.173904	-0.053052	0.333377	0.180284	5.99	0
-1.566983	0.073766	2.399317	0.431354	0.054112	2.026338	-0.411201	0.697581	1.140478	...	-0.144547	0.359667	-0.084026	-0.549121	-0.491289	0.345687	0.138533	0.204048	60.00	0

umns

The 'Amount' variable has a range of 0 to 25,691.16. We use standardisation to eliminate the mean and scale to unit variance to narrow the range so that 68 percent of the results are in the middle (-1, 1).

Test and Train split

y data is taken from the column **Class,** which is response variable and takes value 1 for fraudulent transactions and value 0 for normal transactions.

x data is taken from features **v1,v2,….,v28** which are principal components.

The dataset is split into test and train for the default ratio of **70:30**

Neural Networks

The input neuron, which is the initial layer or input layer, contains each customer's transaction and amount. The hidden layer consists of units and activation function and dropout function. To fine-tune the performance, we can add as many hidden layers as we want. Using Keras' Sequential model, we'll create a 3-layer neural network.

Layer 1*: unit: 1000, activation: relu*
Layer 2*: unit: 1000, activation: relu*
Layer 3*: unit:1, activation:sigmoid*

The number of nodes or neurons in each layer is measured in 'units.' For the hidden layers, we employ the Rectified Linear Unit (ReLU) as an activation function. When it comes to creating deep neural networks, ReLU usually outperforms Sigmoid and Hyperbolic Tangent functions. This is because when the input value is either too large or too small, Sigmoid and Tanh tend to saturate. Furthermore, they only have a strong gradient near their midpoints, such as 0.5 for sigmoid and 0 for tanh. For a binary classification problem, we employ the Sigmoid function in the output layer.

The final layer is the output layer, which is where we get the classified output. The output will either be 1 or 0, with 1 indicating a fraud case and 0 indicating a regular situation.

The model is compiled by providing the optimizer and loss. The train data is fit to the model by mentioning hyperparameters.

Optimizer*:adam*

Loss*:binary_crossentropy*

***Batchsize**:16*

***Epoch**:10*

In the realm of deep learning, Adam' is a prominent algorithm for achieving good results quickly. Every 15 samples, the model weights are adjusted. The full training set is sent through the network once every epoch. Before changing the internal model parameters, a batch specifies how many samples should be iterated through. The predictions are compared to the expected output at the conclusion of the batch, and the error is computed. By going down the error gradient using this error, the optimizer improves the model.

Visualization: -

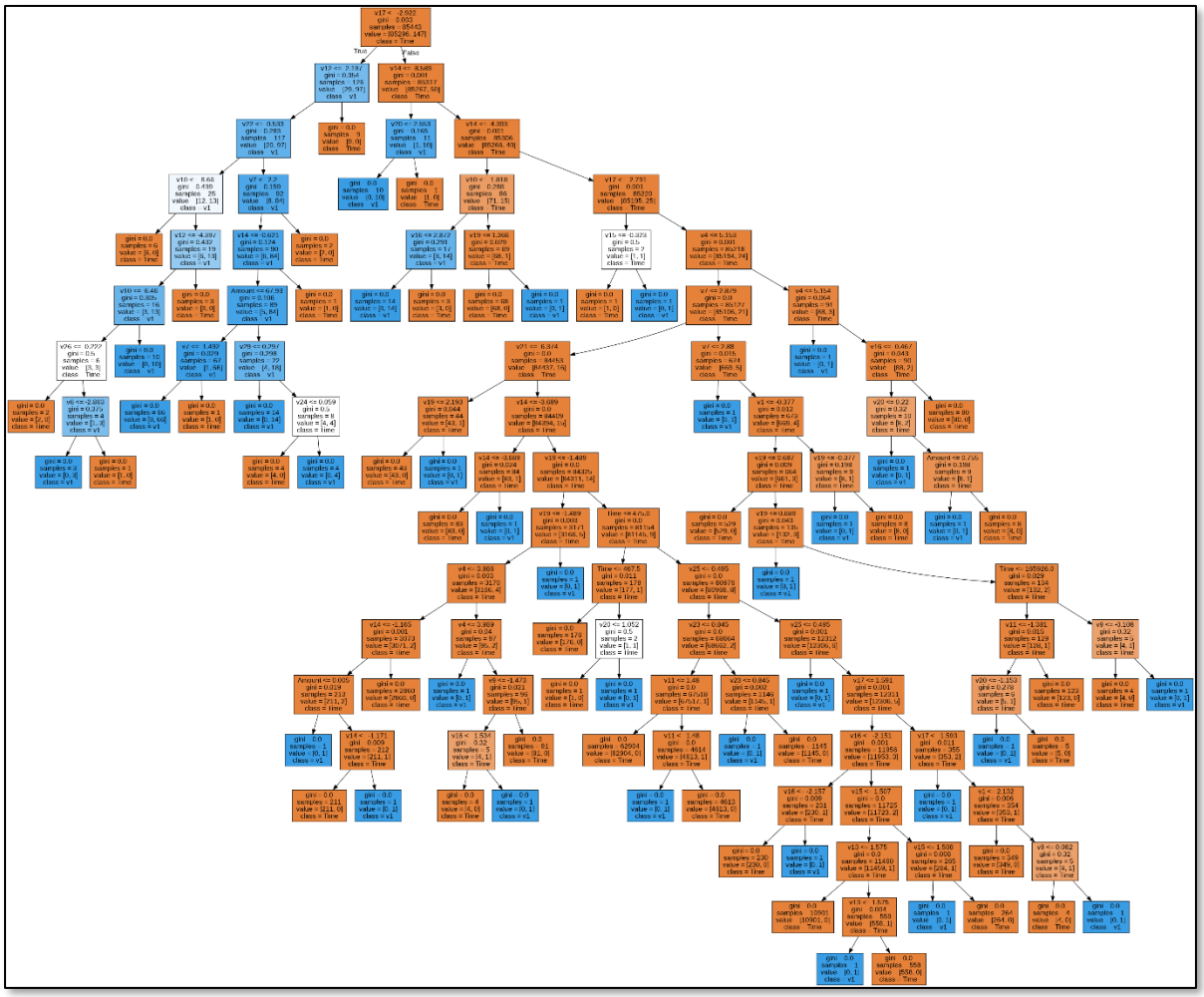
Evaluation metrics:

```
Test loss: 0.021620292216539383
Test accuracy: 0.9982795715332031
```

Decision Tree Classifier:

The same data is fed to build the decision tree model

Visualization:-



Evaluation metrics:

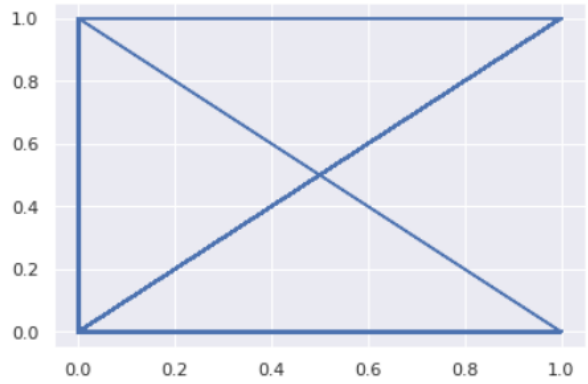
```
Decision Train Accuracy: 0.9990419534118496
Decision Test Accuracy: 0.9993211848834895
--- 26.951398611068726 seconds ---
```

Naïve Bayes Classifier:

The same data is fed to build the gaussian naïve bayes model

Visualization:-

	precision	recall	f1-score	support
0	1.00	0.99	1.00	85296
1	0.15	0.65	0.24	147
accuracy			0.99	85443
macro avg	0.57	0.82	0.62	85443
weighted avg	1.00	0.99	1.00	85443



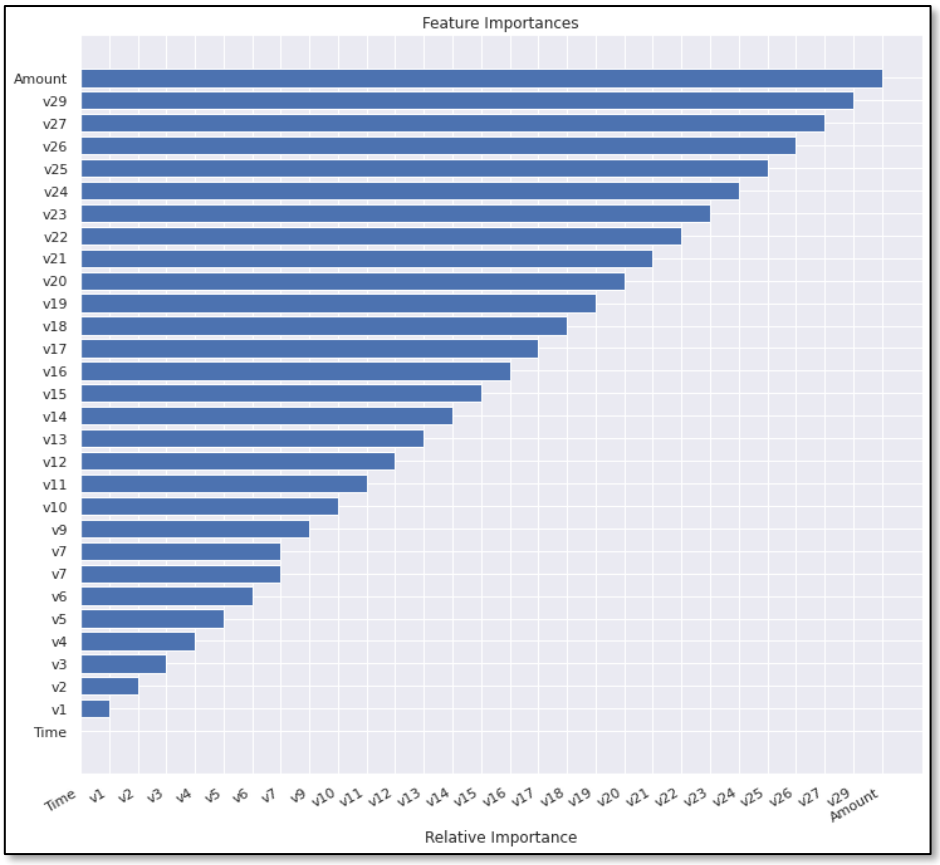
Evaluation metrics:

```
Naive Test Accuracy: 0.9931883389177585
Naive Train Accuracy: 0.9930011820745994
```

Random Forest Classifier:

The same data is fed to build the random forest model

Visualization:-



Evaluation metrics:

```
Accuracy: 0.9995084442259752
```

After Under-sampling:

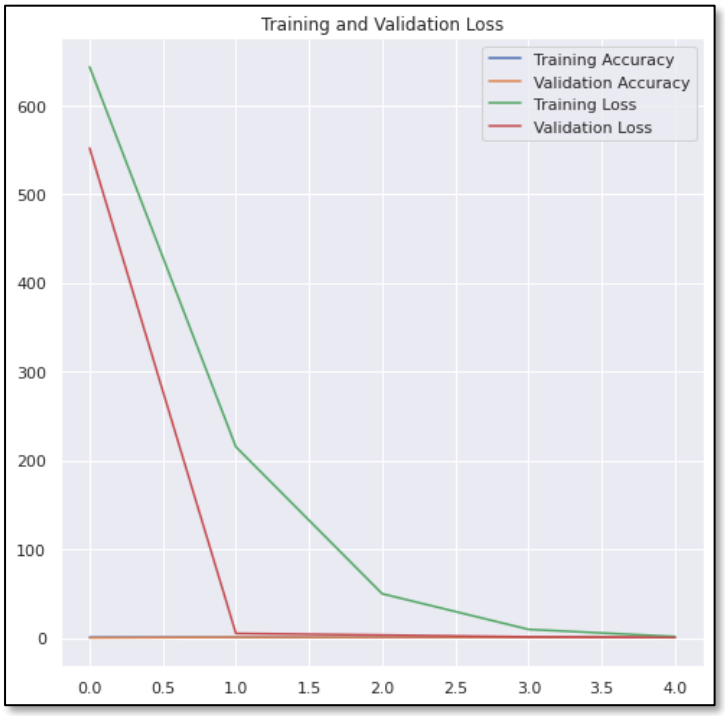
```
Total Number of Normal Data: 2508
Total Number of Fraud Data: 492
```

Neural Networks

1. **Dense Layers:-** 3 -
    - **Activation:-** relu,relu,sigmoid
2. **Drop layers:-** 2
3. **Learning rate:-**=0.01
4. **Optimizer:-** adam
5. **epochs:-**5
6. **batch_size:-**16

u

Visualization:-



Evaluation metrics:

```
Test loss: 0.5286462306976318
Test accuracy: 0.8399999737739563
```

Hyper Parameter Optimization: -

Method 1 :

Hyper Parameter Tunning:-

1. **Dense Layers:-** 5 -
    - **Activation:-** sigmoid
2. **Drop layers:-** 2
3. **Learning rate:-**=0.3
4. **Optimizer:-** Adadelta
5. **epochs:**-20
6. **batch_size:**-16

Visualization:-

Evaluation metrics:

```
Test loss: 0.4724513292312622
Test accuracy: 0.8500000238418579
```

Method 2 :

Hyper Parameter Tunning:-

1. **Dense Layers:-** 5 -
    - **Activation:-** relu, sigmoid, sigmoid , sigmoid , sigmoid
2. **Drop layers:-** 2
3. **Learning rate:-**=0.0
4. **Optimizer:-** adam
5. **epochs:-**20
6. **batch_size:-**16

Visualization:-

Evaluation metrics:

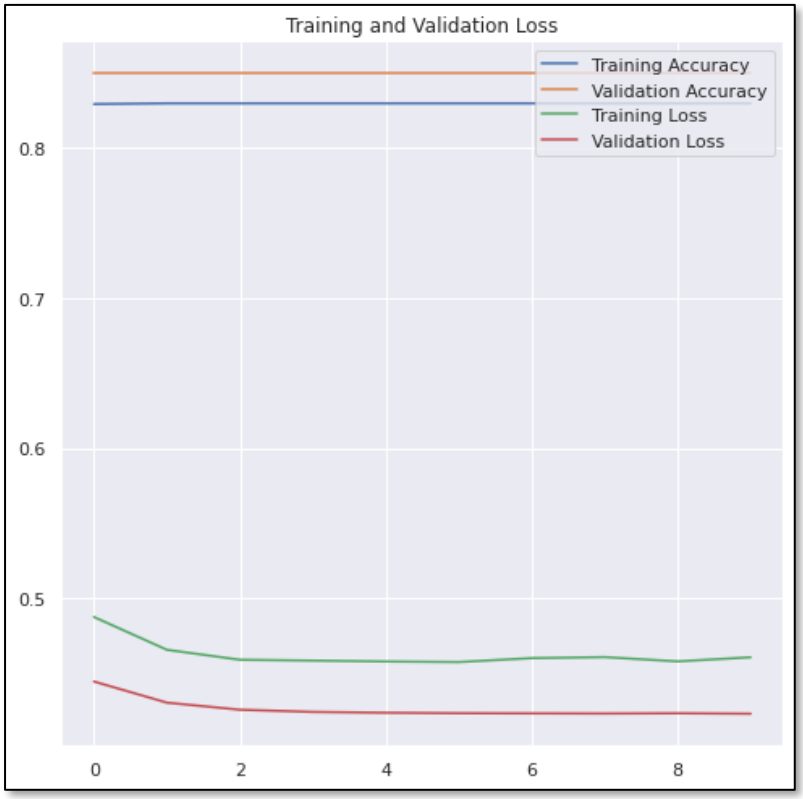
```
Test loss: 0.4237636625766754
Test accuracy: 0.8500000238418579
```

Method 3 :

Hyper Parameter Tunning:-

1. **Dense Layers:-** 3 -
    - **Activation:-** sigmoid , sigmoid , sigmoid
2. **Drop layers:-** 2
3. **Learning rate:-**=0.0
4. **Optimizer:-** Adadelta
5. **epochs:-**10
6. **batch_size:-**16

Visualization:-



Evaluation metrics:

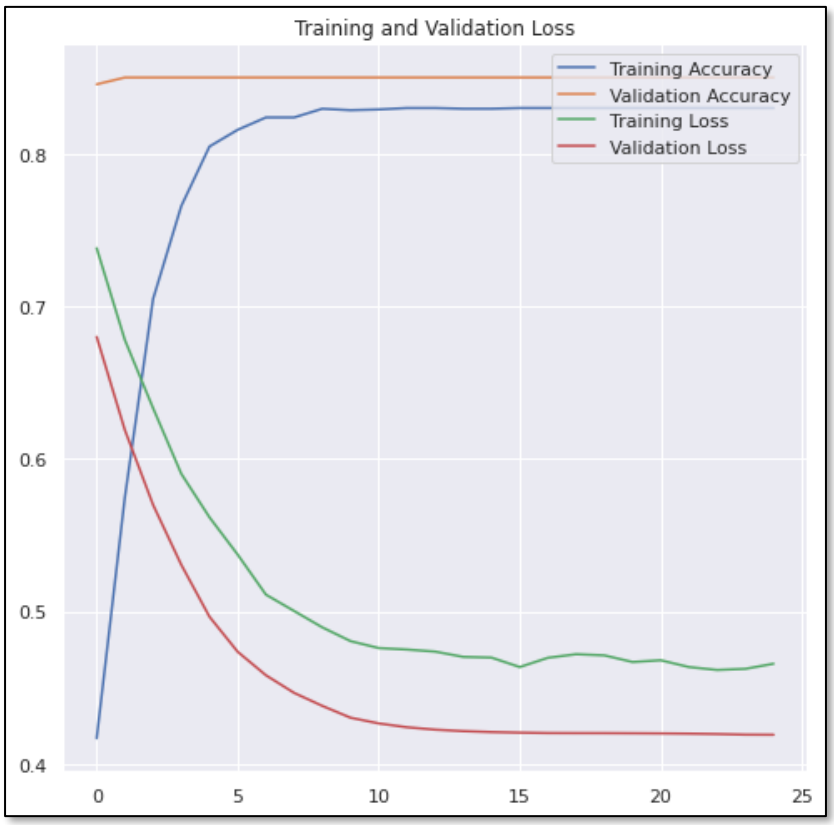
```
Test loss: 0.4230166971683502
Test accuracy: 0.8500000238418579
```

Method 4 :

Hyper Parameter Tunning:-

1. **Dense Layers:-** 5 -
    - **Activation:-** sigmoid , relu , relu,relu,sigmoid
2. **Drop layers:-** 4
3. **Learning rate:-**=0.09
4. **Optimizer:-** Adadelta
5. **epochs:-**25
6. **batch_size:-**17

Visualization:-



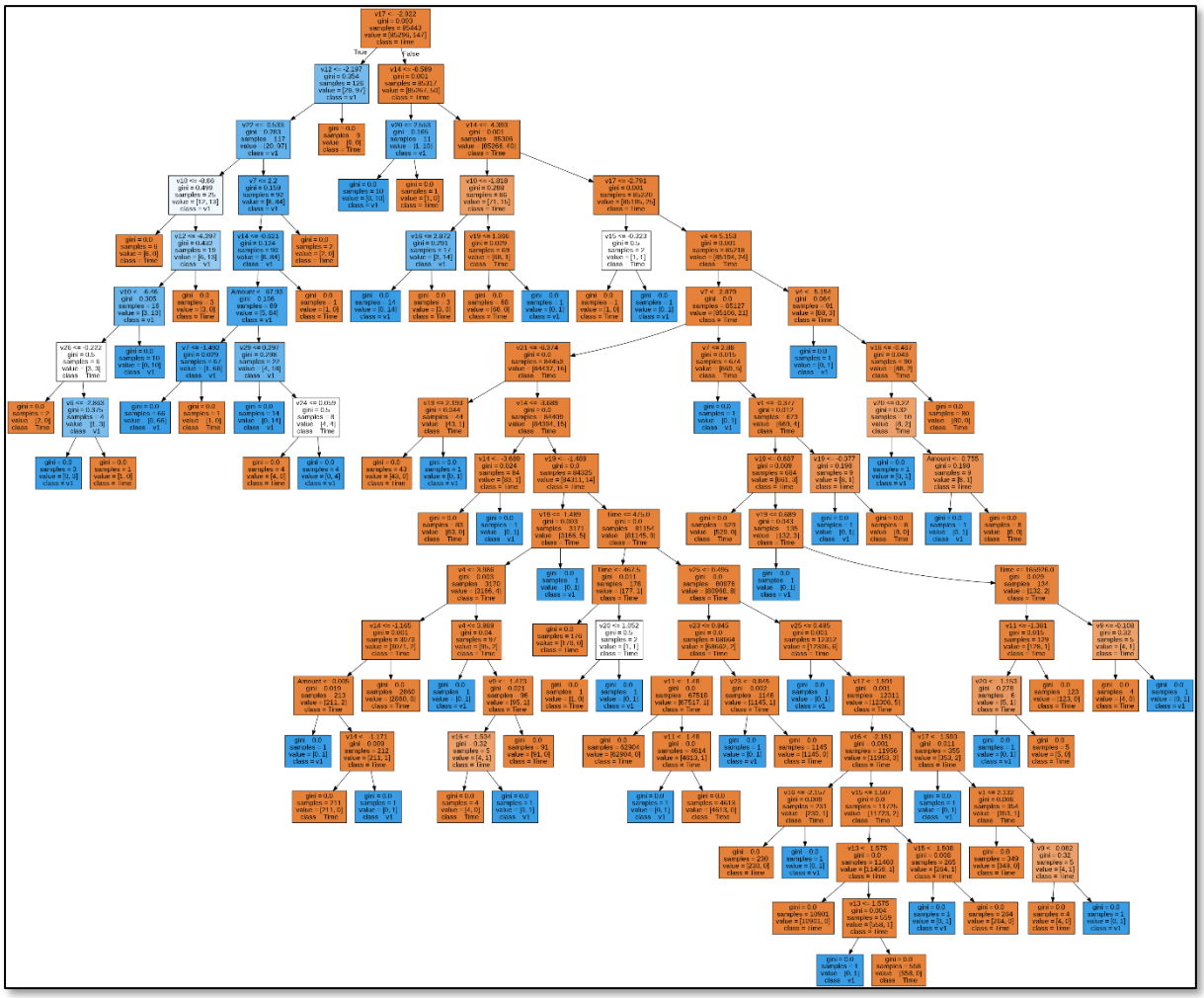
Evaluation metrics:

```
Test loss: 0.4190048575401306
Test accuracy: 0.8500000238418579
```

This would be the maximum accuracy and minimum loss that neural network can achieve.

Decision Tree Classifier:

Visualization:-



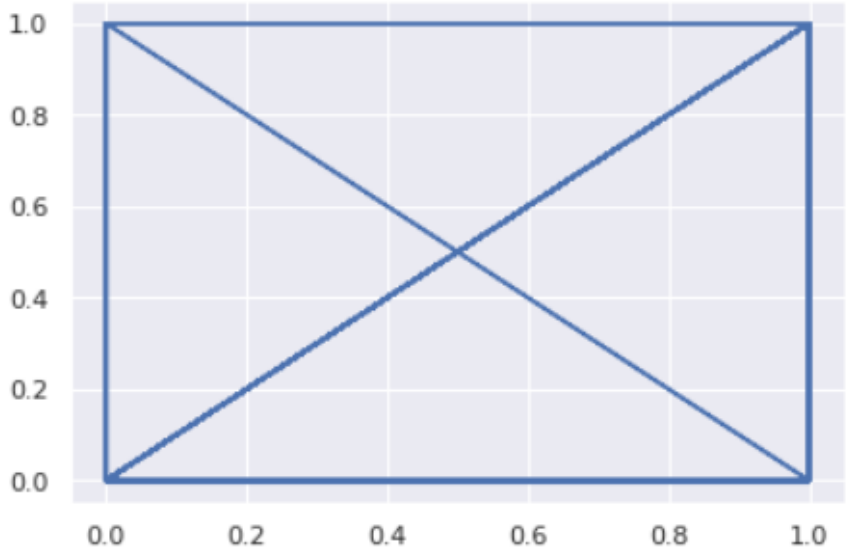
Evaluation metrics:

```
Decision Train Accuracy: 0.9428571428571428
Decision Test Accuracy: 0.9522222222222222
```

Naïve Bayes Classifier:

Visualization:-

	precision	recall	f1-score	support
0	0.95	0.99	0.97	765
1	0.93	0.70	0.80	135
accuracy			0.95	900
macro avg	0.94	0.85	0.89	900
weighted avg	0.95	0.95	0.94	900

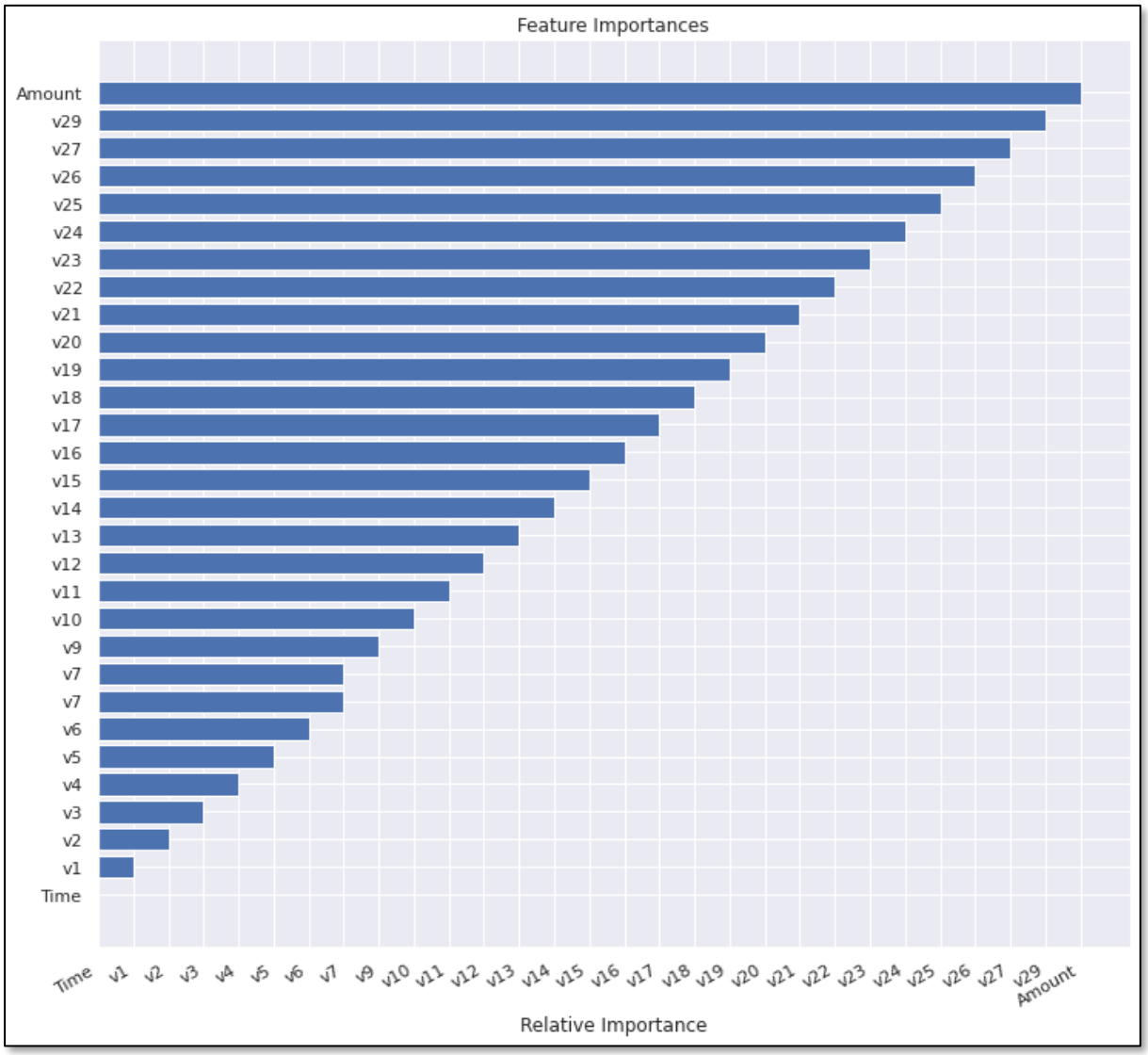


Evaluation metrics:

```
Naive Test Accuracy: 0.94
Naive Train Accuracy: 0.9477777777777778
```

Random Forest Classifier

Visualization:-



Evaluation metrics:

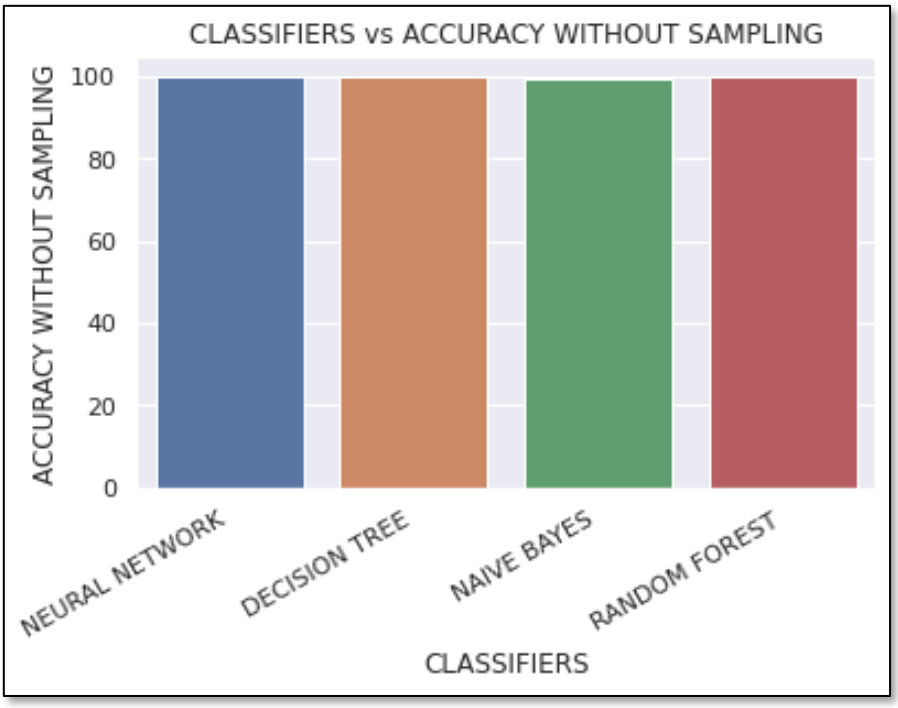
```
Accuracy: 0.9744444444444444
```

Results

Multiple algorithms have been implemented on the same dataset to understand the accuracy and precision of the model.

Before Sampling

Since the dataset was imbalanced with majority of normal transactions compared with fraud (0.17%) transactions. Every model provides an accuracy of 99%. So, if 95% of the data is Negative (non-frauds), the network will cleverly anticipate that all of the data will be Negative, resulting in 95% accuracy. Detecting Positive, on the other hand, is more important than detecting Negative in the case of fraud detection.



After Sampling

Once data set is balanced using under sampling method, it is used to train different models to analyse the accuracy.

The initial Neural network model before hyperparameter optimization provides an accuracy 83.99% with more loss of 52.86%

To minimize the loss and improve the accuracy, hyperparameters are tuned in different ways, explained in 4 methods:

Method 2 focus on increasing the dense layer , learning rate, epoch and optimizer. This provides accuracy of 85% and loss 47.24%

Method 3 focus on combination of activation function & optimizer. This provides accuracy of 85% and loss 42.37%

Method 4 focus on same activation function & changing optimizer. This provides accuracy of 85% and loss 42.30%

Method 5 focus on combination of activation function, changing learning rate, increased epochs and batchsize. This provides accuracy of 85% and loss 41.90%

Hence from all above methods, the neural network can provide maximum accuracy of 85% and minimum loss of 41.90%

METHODS	LAYERS	ACTIVATION	LEARNING RATE	OPTIMIZER	EPOCHS	BATCH SIZE	ACCURACY	LOSS
1	3	relu,relu,sigmoid	0.01	adam	5	16	83.99	52.86
2	5	sigmoid,sigmoid,sigmoid,sigmoid,sigmoid	0.3	adadelta	20	16	85.00	47.24
3	5	relu,sigmoid,sigmoid,sigmoid,sigmoid	n/a	adam	20	16	85.00	42.37
4	3	sigmoid,sigmoid,sigmoid	n/a	adadelta	10	16	85.00	42.30
5	5	sigmoid,relu,relu,relu,sigmoid	0.09	adadelta	25	17	85.00	41.90

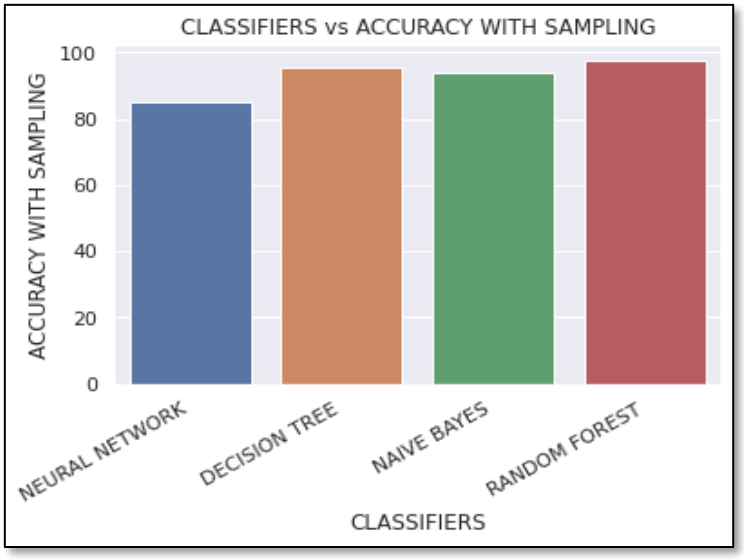
The final neural network model is then compared with the other classifiers like Decision Tree, Naïve Bayes & Random Forest.

The same sampled data is fit to **decision tree model** that provides an accuracy of **95.22%** better than Neural network model

For **Naïve bayes**, the model provides an accuracy of **94%** which is better than neural network but not as decision tree.

For **Random Forest,** the model provides an accuracy of **97.44%** which is better than any of the other models to predict the fraud dataset.

Thus, Random Forest would be the best model to predict the fraudulent credit card transactions with an accuracy of 97.44%

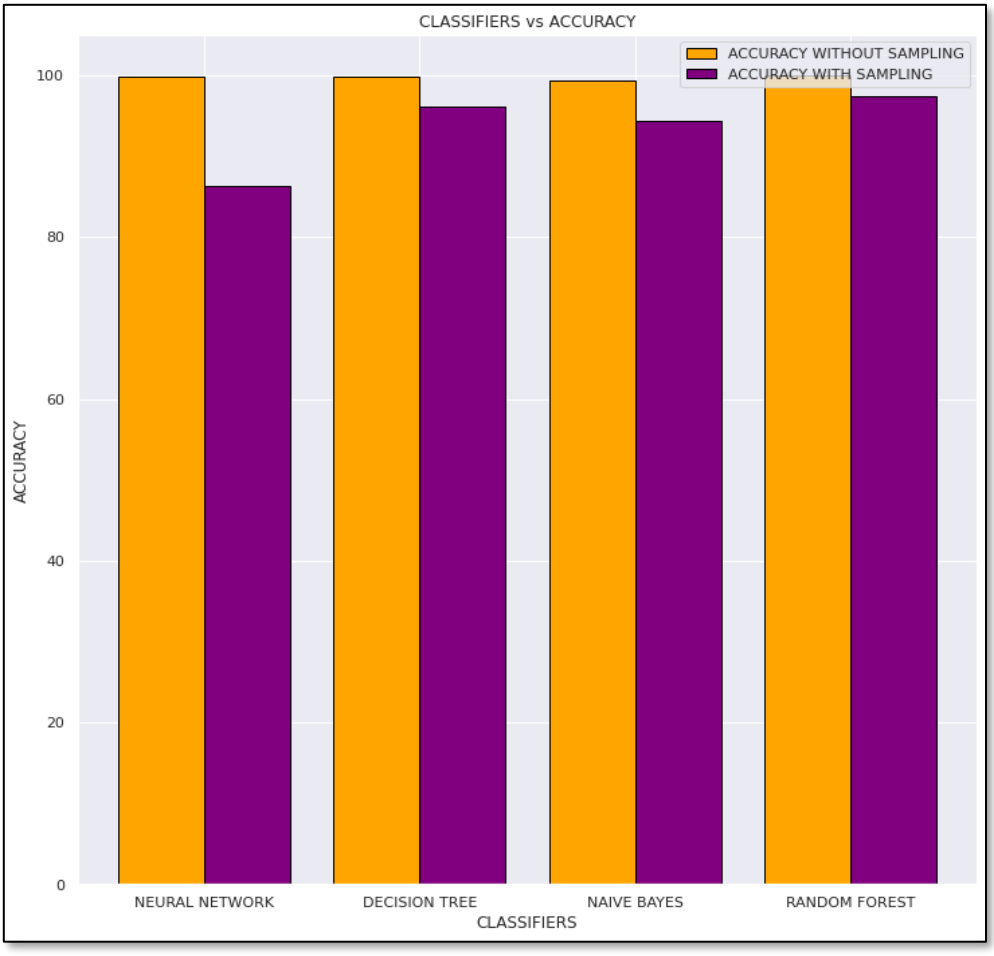


Conclusion

The experimental results show that the Random Forest outperformed other machine learning algorithms, achieving the greatest accuracy of *97.44%*.

In situations where numerous hyper-parameters of an expensive function must be optimised simultaneously and the effective dimensionality is high, future research should examine sequential, adaptive search/optimization techniques.

CLASSIFIERS	ACCURACY WITHOUT SAMPLING	ACCURACY WITH SAMPLING
NEURAL NETWORK	99.82	85.00
DECISION TREE	99.93	95.22
NAIVE BAYES	99.30	94.00
RANDOM FOREST	99.95	97.44



References

[1] Daniel.Hyperparameter Importance Across Datasets.2019:
https://arxiv.org/pdf/1710.04725.pdf

[2] Bergstra.Random Search for Hyperparameter Optimization. 2012.
http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

[3] Jaderberg .Population Based Training of Neural Networks. 2017.
https://arxiv.org/pdf/1711.09846.pdf

[4] Breiman. Random Forests. Machine learning 45, 1 (2001), 5–32. 2001

[5] Brijain R Patel. A Survey on Decision Tree Algorithm For Classification.2014

[6] Irina Rish.An empirical study of the naive Bayes classifier.2001
https://www.researchgate.net/publication/228845263_An_Empirical_Study_of_the_Naive_Bayes_Classifier

[7] Maniraj.Credit Card Fraud Detection using Machine Learning and Data Science.2019
https://www.ijert.org/research/credit-card-fraud-detection-using-machine-learning-and-data-science-IJERTV8IS090031.pdf

[8] Kartik. Credit Card Fraud Detection System.2021
https://www.ijrte.org/wp-content/uploads/papers/v10i2/B62580710221.pdf