A Comparative Analysis of Credit Card Fraud Detection Using Machine Learning Classification Algorithms

By

M. Shahrear Shohag

152-19-1783

Supervised By

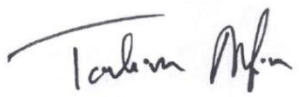
Md. Mushfiqul Islam
Lecturer
Department of ETE
Faculty of Engineering
Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY
DHAKA-1207, BANGLADESH

APPROVAL

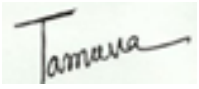
This thesis titled **"A Comparative Analysis of Credit Card Fraud Detection Using Machine Learning Classification Algorithms"** submitted by M. Shahrear Shohag the Department of Electronics and Telecommunication Engineering (ETE), Daffodil International University, has been accepted as satisfactory for the partial fulfillment of the requirements for the bachelor degree. in Electronics and Telecommunication Engineering and approved as to its style and contents.

BOARD OF EXAMINERS



Associate Prof. Md. Arefin Taslim **Chairman**
Head
Department of ETE
Faculty of Engineering
Daffodil International University

Professor Dr. A K M Fazlul Haque **Internal Examiner**
Professor & Associate Dean Faculty of Engineering
Daffodil International University



Ms. Tasnuva Ali **Internal Examiner**
Assistant Professor
Department of ETE
Faculty of Engineering
Daffodil International University

Dr. Saeed Mahmud Ullah **External Examiner**
Associate Professor
Department of Electrical and Electronics Engineering University of Dhaka

DECLARATION

I hereby declare that this research is my own work and effort under the supervision of **Md. Mushfiqul Islam, Md. Mushfiqul Islam, Department of ETE, Faculty of Engineering,** Daffodil International University, Dhaka. It has not been submitted anywhere for any award. Where other sources of information have been used, they have been acknowledged.

Supervised By:

Md. Mushfiqul Islam
Lecturer
Department of ETE
Faculty of Engineering
Daffodil International University

Submitted By:

M. Shahrear Shohag
152-19-1783
Department of ETE
Daffodil International University

DEDICATION

This thesis is wholeheartedly committed to my dearest guardians, who have been our source of motivation and gave us strength when we thought of surrendering, who consistently provide their ethical, spiritual, feeling and financial support.

ACKNOWLEDGEMENT

First and foremost, we would like to express and convey our gratitude to the Almighty Allah,for his blessing approval, protection, mental power and wisdom in all aspect of our life, and all applause to Allah of Complete this thesis.

The real sprit of achieving a goal is through the way of excellence and austere discipline. We would have never succeeded in completing our task without the cooperation, encouragement and help provided by various personalities.

This work would not have been possible without the support and guidance of Md. Mushfiqul Islam, Lecturer, Department of ETE, Faculty of Engineering, Daffodil International University, Dhaka, who happen to be my research supervisor for his endless patience, scholarly guidance continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading inferior drafts and correcting them at all stages have made it possible to complete this research. And we choose this topic and developed the research. Deep Knowledge and keep interest of our supervisor in the field of Biomedical and influenced us to carry out of this research.

We would like to express our heartiest gratitude to all of our faculty members, Department of Electronics and Telecommunication Engineering, for their kind help to finish our thesis and also we would like to thank the staffs of the ETE Department of Daffodil International University.

We must sincere to acknowledge with due respect the constant support and patience of our family members and outfriends their helpful and assistance of completing our program.

M. Shahrear Shohag

ABSTRACT

With the rise of e-commerce and online purchases in the modern age, credit card fraud has become a severe and growing issue. Such unpleasant practices can impact millions of people around the world through this identity theft and the loss of money. Crime is a growing threat with far-reaching consequences to the finance sector. The extraction of information seemed to be a core job for payment fraud recognition, fraud detection efficiency in card buy-outs has a significant impact on the measurement strategy for the data set, the choices of the variable and the techniques of detection used. We'll aim at how Artificial Neural Networks (ANN), Decision Trees, K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machines (SVM), Random Forests, Neural Network Supervised (MLPClassifier), Ridge Classification, AdaBoost Classification, and Naive Bayes are implemented in this research. Classification algorithms for highly skewed credit card fraud results. For model understanding, accuracy, f1, recall, precision, Matthew's correlation coefficient (MCC), confusion matrix, and lime which will be used to evaluate the execution of these techniques.

Table of Contents

List of Table

List of Figures

Chapter 1
Introduction

1.1 Introduction

Financial fraudulent activity in an ever growing threat in this modern era, where business, industries, organizations and government is highly dependable on the internet technology. And with rapidly developing the economy globalization, credit card transactions became popular for commercial transactions in both online and offline. According to Global Payment report 2020, 24.2% in eCommerce and 20.9% in Point of Sale (POS) payment had made by credit card in 2019. Thus, the huge transactional services often target by cyber criminals to perform fraudulent activity using credit card services.

The unauthorized usages of card and unusual transactions are defined as a credit card fraud. Namely there are three three types of credit card frauds in general. Conventional frauds, which performed by stilling, faking or counterfeiting. Online frauds are often committed by the fake or false merchant sites. And Merchant related frauds involved with merchant collusion and triangulation.

By using traditional methods of manual detection can be time consuming, inefficient and costly for detecting the fraudulent transactions. Credit card fraud detection is the process of identifying transaction whether it's legitimate (genuine) or fraudulent. To counter cyber-criminal activities, data mining and machine learning approaches are vastly used. Therefore, these techniques can be applied for credit card fraud detection methods.

1.2 Aim and Objective

- Find an appropriate deep neural architecture for the fraud detection method, based on accuracy.
- Find an appropriate agent architecture for the fraud detection system based on accuracy.
- Develop and incorporate a framework in the available database to identify fraudulent transactions where the number of cases of fraud is very limited compared to legal cases.
- Determine which algorithm gives a better result based on accuracy, Recall and Precision

1.3 Limitations and Challenges

- Huge amount of data processing everyday.
- Build model have to fast enough to respond to the scam in time.
- Maintain high accuracy.
- Imbalanced Data (i.e. mostly 99.8% are legitimate or non-fraudulent which makes it hard to direct the fraudulent one).
- Data unavailability as most are private.
- Another major issue can be misclassified data, as not every single fraudulent transaction is caught or reported.
- Adaptivity against scammers.

1.4 Chapter Description

- **Chapter 2 - Technical Background** shows the background of Machine Learning approaches discussed in this work.
- **Chapter 3 - The Data Overview** explains the database and analyzes it.
- **Chapter 4 – Data Sorting and Exploration,** preparing the data to test, train and predict.
- **Chapter 5 – Modeling** with ten classification model (ANN, Decision Tree, KNN, Logistic Regression, SVM, Random Forest, Neural Network Supervised (MLPClassifier), Ridge Classification, AdaBoost Classification and Naive Bayes).
- **Chapter 6 – Model Evaluation** with accuracy, f1, recall, precision, Matthew's correlation coefficient (MCC),confusion matrix and used lime for model interpretation.
- **Chapter 7 - Results** provides the provisional and final results of the study conducted.

Chapter 2
Technical Background

In this chapter introduces each technique and definition used in the creation of work in the field of machine learning and deep learning. These terms, beginning with , Artificial Neural Network (ANN), Decision tree , K-nearest Neighbors (KNN), Logistic Regression algorithm, Support Vector Machine(SVM), Random Forest Tree Algorithm and Naive Bayes Classifier, are clearly explained, trying to make them easy to understand.

2.1 Artificial neural network

Artificial neural networks (ANNs) are computation systems largely inspired by the biological neural networks that form animal brains, often called as neural networks (NNs).

ANN were developed as a mathematical generalization of the components of the human brain, specifically neuron networks that obtain information to learn characteristics and take action according to the ANN objective. Artificial neural networks find the best combination of parameters that match a given problem during the training phase. The study of artificial neural networks is inspired by their similarities to biological systems that function efficiently, consisting of very basic yet multiple nerve cells that work massively and have the ability to learn in parallel. After good training, an ANN may find logical solutions to similar problems in the same class that have not been specifically trained, since they have the capacity to generalize and associate knowledge. In exchange, this results in a high level of fault tolerance against noisy data inputs.

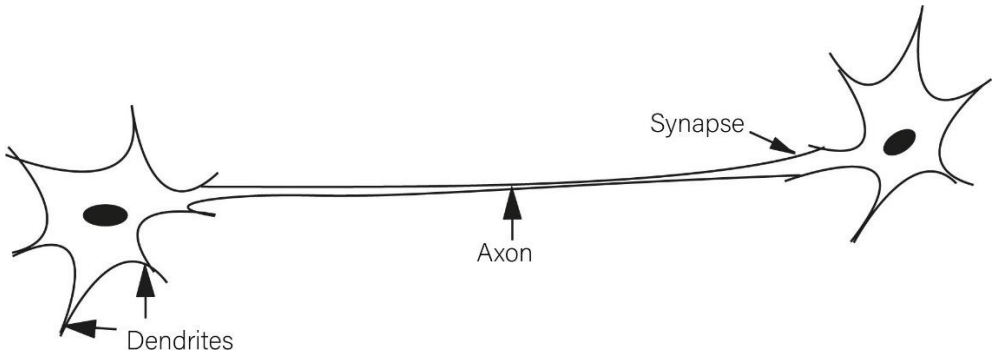


Figure 2. 1 Natural Neurons

That of natural neurons is the model that remains. A basic drawing of a biological neuron is seen in Figure 2.1. Via synapses located on the neuron dendrites or membrane, natural neurons receive signals. The neuron is triggered when the signals received are high enough and emits a signal through the axon. This signal might be sent to another synapse, and other neurons might be activated. If we equate them with artificial neurons, natural neurons have immense complexity. However, when designing our neural networks, we can extract features that are important. Inputs are needed, multiplied by initial weights, to be enabled by a mathematical function afterwards. An example of an artificial neuron is shown in Figure 2.2.

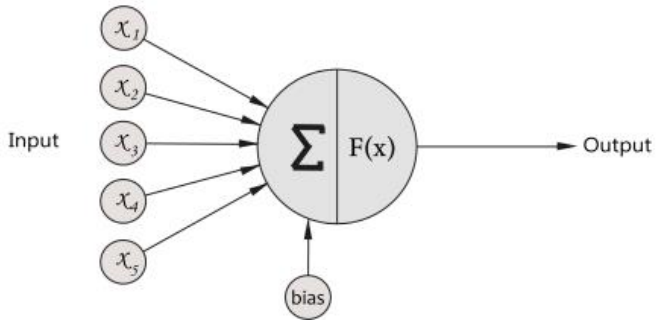


Figure 2. 2
Artificial Neuron

Artificial Neuron

An artificial neuron is a natural neuron inspired computational model. The architecture and functioning of the biological neuron, which is the basic building block of biological neural networks, including the brain, spinal cord and peripheral ganglia, is derived from observation. It is possible to research an artificial neuron using a simple mathematical procedure. Project, such as the following:

$$y(k) = F\left(\sum_{i=0}^{m} \omega_i(k).x_i(k) + b\right) \quad 2.1$$

Where,

- $\boldsymbol{x_i}$**(k)** is the discrete time k input
- $\boldsymbol{\omega_i(k)}$ is the discrete time k weight value
- **b** is bias
- **F** is a transfer function
- $\boldsymbol{y_i(k)}$ is the discrete time k output value

Loss Function

To minimize the so-called loss on the training set is the standard approach for training a deterministic regression or classification model. The loss is defined as a function of the parameters of the model θ:

$$l(\theta) = \frac{1}{S}\sum_{s=0}^{S} L(y_\theta(X_s), T_s) \quad 2.2$$

WhereX_s is a sample of training with corresponding goal T_s and $y_\theta(x)$is the prediction of the input x model given. Measure of Loss $L : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ To each sample, R assigns a loss value based on the discrepancy between the estimation of the model and the ground truth. The function of loss is used to assess and minimize the difference between expected

Real performance and. Mean Squared Error (MSE) is one of the most used loss functions:

$$MES(x, \hat{x}) \frac{\sum_{i=1}^{N} \| x_i - \hat{x_i} \|^2}{N} \quad 2.3$$

Where x is the target and $\hat{x}$ is the obtained value.

Transfer Function

In the 2.1 equation, the transfer function is one of the most important variables. Another term by which transfer functions are named is Activation Functions. Activation functions are functions used to determine the weighted total of inputs and biases in neural networks, which are used to assess whether or not a neuron can be shot. The Artificial Neural Network properties are defined by these functions. There are several types of activation functionality that are used according to the issue that needs to be addressed. The graphs of the most general functions are shown in Figure 2.3 (retrieved from) and are defined as:

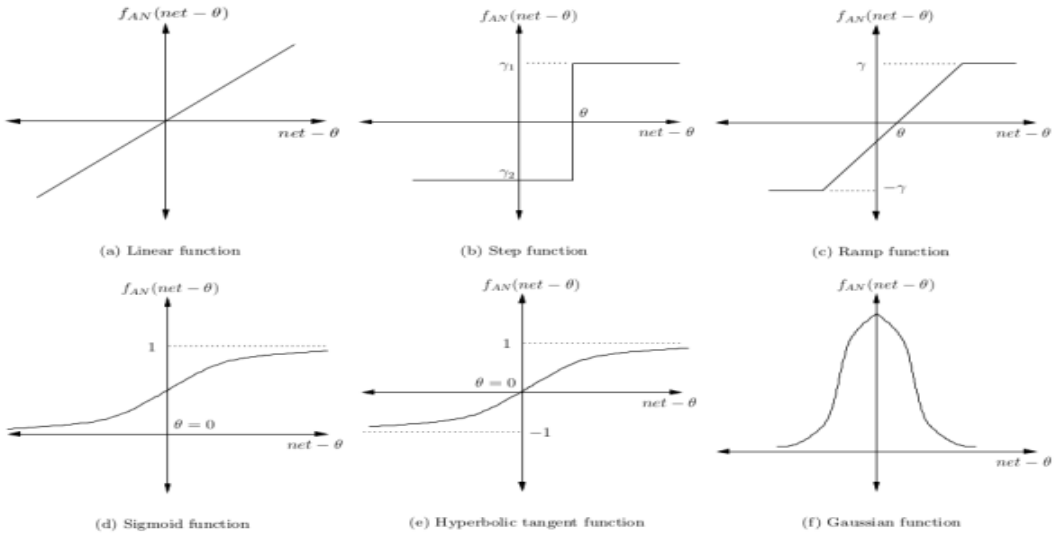


Figure 2. 3 Transfer Function

Linear Function

The input/output properties of most actual models are non-linear. But there are several models that have conduct that is similar enough to linear if run within nominal parameters. In these types of cases, this feature may be an appropriate reflection of input/output actions. There are no thresholds applied by the linear function and the output is similar to the input.

$$f(x) = x \quad 2.4$$

Step Function

This is used to model the behavior of the classic 'All-or-none.' It resembles a ramp function, but when a threshold value is reached, the function value θ shifts abruptly.

$$f(x) = \begin{cases} 0, & if\ x \leq \theta \\ 1, & if\ x > \theta \end{cases} \quad 2.5$$

Ramp Function

With the Linear output function, the Ramp function combines the Step function. The neuron exhibits the output f(x)= 0 as long as the activation is smaller than the threshold value θ_1, if the activation reaches the threshold valueθ_2, the output is f(x)= 1. The activation output of the neuron at the interval between the two threshold values of $\theta_1 < x < \theta_2$ is calculated by a linear activation interpolation.

$$f(x) = \begin{cases} 0 & if\ x \leq \theta_1 \\ \dfrac{x - \theta_1}{\theta_2 - \theta_1} & if \theta_1 \leq x \leq \theta_2 \\ 1 & if\ x > \theta_2 \end{cases} \quad 2.6$$

Sigmoid Function

The input is taken and compressed the output by This function of transfer in the range 0 to 1. In multilayer networks that are trained using the Backpropagation Algorithm1, this transfer function is widely used, in part because this function is distinguishable in its entire range. This function is defined mathematically as,

$$f(x) = \frac{1}{1 + \epsilon^{-x}} \quad 2.7$$

Hyperbolic Target Function

In terms of neural networks, this function is comparable to a bipolar sigmoid that has an output range from −1 to +1. For neural networks, where speed is more important than the exact shape of the transfer function, this function is a reasonable trade-off.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad 2.8$$

Gaussian Function

For zero activation, a Gaussian function's maximum function value is found. Even the function is: f(−x) = f (x). The value of the function decreases as the absolute activation value increases.

$$f(x) = e^{-x} \quad 2.9$$

Learning Rate

The learning rate is a parameter in neural networks that defines how much the weights will adjust in response to an observed mistake on the training set. Choosing this learning rate can have a dramatic impact on the precision of generalization as well as the pace of training. This value is a proportionality constant that configures the scale of the weight changes. The value of this constant typically varies throughout the [0; 1] interval. If the learning rate is too high, the average loss will rise and get stuck at or even diverge from a local minimum. If the value is too poor, however, the learning rate can lead to slow convergence.

Epochs

When the set of training is finite, training continues by sweeps through the set of training called an age, and complete training typically takes several epochs (iterations through the set of training). For example, for each model (i.e.) a network with a different set of weights, the back propagation learning algorithm constructs a different model. The learning algorithm examines or travels through 1000 different models if a neural network is trained for 1000 epochs.

2.1.1 Learning Paradigms of ANN

There are more and more types of ANNs that refer to a variety of fields. Therefore, to enable their use, they must be categorized correctly. There are many ways of categorizing them, such as the type of transfer functions, the topology, implementations, algorithm type, etc. A brief classification according to the learning paradigms is shown in this section. Learning may refer to either knowledge acquisition or enhancement. The learning process is a method for updating an ANN's architecture and link weights to maximize its effectiveness in order to perform a particular task. The following are the three primary learning paradigms: supervised, unsupervised (or self-organized), and reinforcement. Numerous algorithms include each category.

Supervised Learning

In supervised learning, in the form of the exact activation of all output neurons, the training set consists of input patterns and their correct outcomes. Then, each output produced by the training set is compared with the correct solution (target) and the neural network's synaptic weights are modified according to this comparison. The main aim of this training is to change the weights so that the difference is minimal between performance and target. The Error Backpropagation Algorithm usually solves learning by practicing in a supervised ANN model.

Supervised learning is a common technique since it facilitates the ability of neural networks to generalize, that is, even with new data without prior knowledge of the goal, to give correct results. Normally, this method of learning is used for classification for which there are several choices for each problem type. However, choosing an appropriate classifier (Multilayer Perceptron, Support Vector Machines, K-nearest Neighbors Algorithm, Gaussian Mixture Model, Gaussian, Naive Bayes, Decision Tree, Radial Basis Function Classifiers,etc.) is still

more art than science for a given issue. A general representation of Supervised learning paradigm is shown in figures 2.4

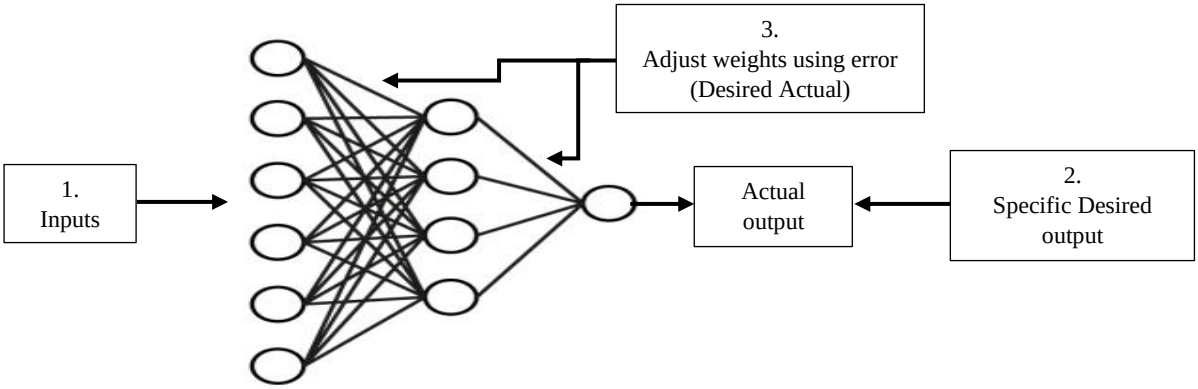


Figure 2. 4 Supervised Learning

Unsupervised Learning

The most biologically plausible method is unsupervised learning, but it is not ideal for all problems. Only the input patterns are given; the network attempts to recognize and classify similar patterns into similar categories. Neural networks that are trained using unsupervised methods are referred to as self-organizing because they are not driven by what should be the optimal or right performance. The output processing units self-organize when faced with a set of input patterns by initially competing to identify the pattern, and then coordinating to change their relation weights.

Unsupervised learning is mainly used in applications such as statistical modeling, compression, filtering, blind source separation and clustering that fall within the field of estimation problems. The last one is a typical method of unsupervised learning where we try to categorize data by similarity in different clusters. Self-organizing maps are the ones that use unsupervised learning algorithms most often. A general representation of Unsupervised learning paradigm is shown in figure 2.5

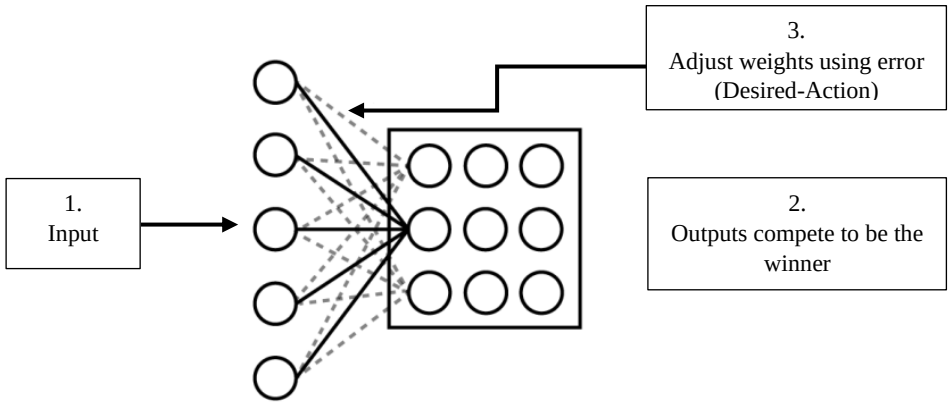


Figure 2. 5 Unsupervised Learning

Reinforcement Learning

The training set consists of input patterns, a value is returned to the network after a sequence is completed indicating whether the outcome was correct or wrong and, likely, how right or wrong it was. In order to learn an optimal (or nearly optimal) environmental policy, the goal of reinforcement learning is to use observed rewards. Reinforcement learning is learning through engaging with an environment by taking multiple actions and witnessing several

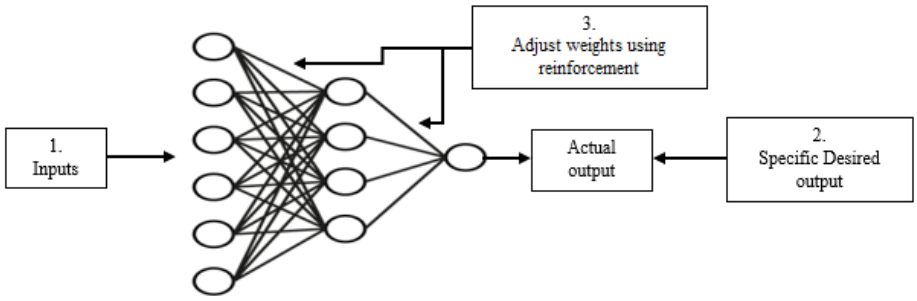


Figure 2. 6 Reinforcement Learning

defeats and accomplishments while attempting to increase the rewards obtained. Agent 2 is not aware of what action to take. Reinforcement learning is especially suited to issues like a trade-off of long-term and short-term reward. It has been successfully applied to various issues, including control of robots, telecommunications, and games such as chess and other sequential tasks of decision-making. A general representation of Reinforcement Learning learning paradigm is shown in figures 2.6.

2.1.2 Common problems and Methods of data splitting

Two common problems can arise in the training of a neural network, which can be avoided by choosing a proper method of data splitting. Those issues are:

- **Underfitting** occurs when there is very little training information provided to the neural network and therefore does not allow learning to be generalized. That is, the model has not learned enough, resulting in low generalization and results that are unreliable.
- **Overfitting** is a key issue in machine learning tasks that are supervised. If a learning algorithm fits the training data set so well that noise and the peculiarities of the training data are memorized, it is the phenomenon identified. This problem leads to the deterioration of the model's generalization properties and, when applied to new measurements, results in its unreliable performance.

There are many splitting techniques that can be used, but dividing the data into three subsets is one of the most common.

- **Training**: the knowledge used to teach (train) the algorithm to perform its assignment.
- **Validation**: the data used to tune a learning algorithm's hyperparameters.
- **Testing**: data used to validate the behaviors of the machine learning model.

It is a task that requires many tests to determine the division of data that goes to each subset, which then produces the best model for each problem. There are many suggestions, though, such as dividing or even using empirical tests manually.

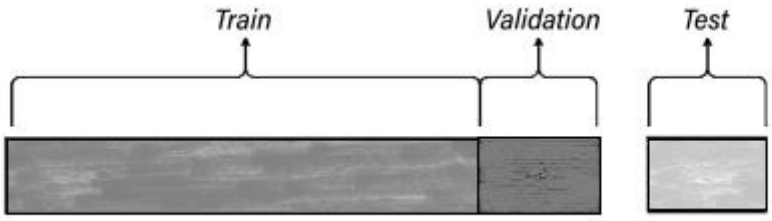


Figure 2. 7 A data splitting visualization

2.2 Random Forest

Random forests or random decision forests are a supervised learning algorithm for classification, regression and other tasks that function by creating a number of decision trees at training time and generating the class that is the class mode (classification) or the individual trees' mean/average prediction (regression). Random forests of decision making correct the practice of overfitting their training set for decision trees. However, their performance may be influenced by data characteristics.

There are a number of applications for Random Forests, such as recommendation engines, classification of images and selection of features. It can be used to categorize loyal applicants for loans, classify fraudulent behavior and predict diseases. It lies at the basis of the Boruta algorithm, which in a dataset selects important features.

It is theoretically an ensemble technique of decision trees generated on a randomly divided dataset (based on the divide-and-conquer approach). This set of classifiers for the decision tree is also known as the forest. The individual decision trees are created for each attribute using an attribute selection indicator such as data gain, gain ratio and Gini index. Each tree depends on a random sample that is independent. Each tree votes and the most common class is selected as the final outcome in a classification problem. In the case of regression, the final outcome is called the sum of all the outputs of the tree. In comparison to the other non-linear classification algorithms, it is simpler and more efficient.

Preliminaries: decision tree learning

For different machine learning tasks, decision trees are a common tool. In particular, very deeply grown trees appear to learn extremely erratic patterns: they overfit their training sets, i.e., they have low bias, but very high variance. Random forests are a way to average many deep decision trees, trained with the intention of reducing the variance on various sections of the same training set. This comes at the cost of a slight increase in bias and some lack of interpretability, but usually improves the performance in the final model significantly.

Forests are like pulling decision tree algorithm attempts together. In this way, the teamwork of multiple trees increases the effectiveness of a single random tree. Forests do offer the results of a K-fold cross validation, but not very similar.

Bagging

The training algorithm for random forests applies to tree learners the general bootstrap aggregating method, or bagging. Provided the training set X = x_1, ..., x_n with answers Y = y_1, ..., y_n, bagging repeatedly (B times), a random sample is chosen to replace the training set and matches the following samples with trees:

For b = 1, ..., for B:

1. Test, n training examples from X, Y with replacement; call these X_b, Y_b.
2. Train f_b on X_b, Y_b, to a classification or regression tree.

Predictions for unseen samples x' can be made after training by averaging the predictions on x' from all the individual regression trees:

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x') \qquad 2.10$$

In the case of classification trees, or by taking the majority vote.

This bootstrapping technique leads to better performance of the model because, without increasing the bias, it reduces the variance of the model. This implies that while a single tree's predictions in its training set are extremely sensitive to noise, the average of several trees is not, as long as the trees are not correlated. It will give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic) to simply train many trees on a single training set; bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

In addition, the uncertainty of the forecast can be calculated as the standard deviation of the forecasts from all the individual regression trees on x':

$$\sigma = \sqrt{\frac{\sum_{b-1}^{B}\left(f_b(x') - \hat{f}\right)^2}{B-1}} \qquad 2.11$$

The sample/tree count, B, is a free parameter. A few hundred to several thousand trees, depending on the size and design of the training collection, are usually used. Using cross validation or observing the out-of-bag error, the optimal number of trees B can be found: the mean prediction error on each training samplex_i, using only the trees that did not have x_i in their bootstrap sample. After certain numbers of trees have been fit, the training and test error appears to level off.

From bagging to random forests

The original bagging algorithm for trees is defined in equation 2.10. Random forests vary from this general scheme in just one way: they use a modified tree learning algorithm that selects a random subset of features for each candidate split in the learning process. This process is called "feature bagging" sometimes. In an ordinary bootstrap sample, the explanation for doing this is the correlation of the trees: if one or a few characteristics are very good predictors for the response variable (target output), these characteristics will be selected in many of the B trees, allowing them to become correlated. Ho offers an overview of how bagging and random subspace projection lead to precision gains under various conditions.

Usually, in each break, $\sqrt{p}$ (rounded down) characteristics are used for a classification problem with p functions. The inventors suggest $p/3$ (rounded down) for regression problems with a minimum node size of 5 as the norm. The best values for these parameters in practice would depend on the issue and should be viewed as tuning parameters.

ExtraTrees

Adding another randomization stage yields highly randomized trees, or ExtraTrees. Although similar to ordinary random forests, there are two key differences in that they are an ensemble of individual trees: first, each tree is trained using the entire learning sample (rather than a bootstrap sample), and second, the top-down splitting is randomized in the tree learner. A random cut-point is chosen instead of calculating the locally optimal cut-point for each function under consideration (based on, e.g., information gain or Gini impurity). This value is chosen from a uniform distribution within the empirical range of the function (in the training set of the tree). Then, to split the node, the split that yields the highest score is chosen from all randomly generated splits. The number of randomly selected features to be considered at each node can be defined, similar to ordinary random forests. The default values for this parameter are $\sqrt{p}$ for classification, and p for regression, where the number of features in the model is p.

2.2.1 Working Function of Random Forest Algorithm

By following steps, we can comprehend the working Function of the Random Forest algorithm.

- Start by selecting random samples from a given dataset first.
- Next, for each sample, this algorithm will create a decision tree. Then, from any decision tree, it will get the prediction result.
- For any predicted outcome, voting will be carried out in this step.
- Eventually, pick the outcome of the most voted prediction as the final result of the prediction.

The diagram below illustrates the Random Forest algorithm working function:

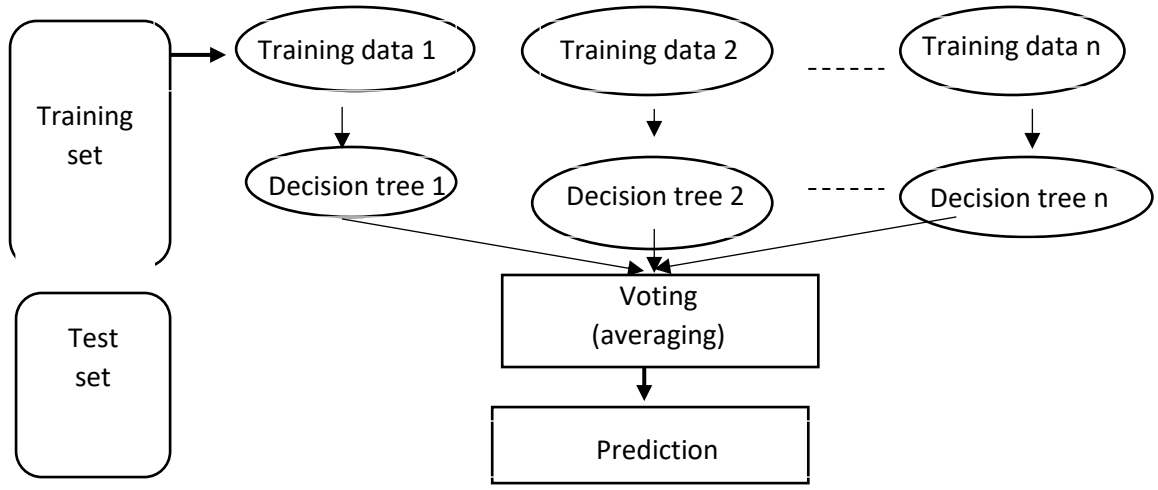


Figure 2. 8 Illustrations of Random Forest algorithm working function.

2.3 K-nearest Neighbor (KNN)

The k-nearest neighbor algorithm (KNN) is a method of non-parametric classification in statistics. It is used for regression and classification. In both instances, the input consists of the examples of k training nearest to the dataset. The performance depends on whether the classification or regression uses k-NN.

The performance is a class affiliation in the KNN classification. An object is graded by its neighbors' majority of votes, with the object being allocated to the most common class amongst the nearest neighbors. Where k is typically a small positive integer. If k = 1, therefore the component is simply assigned to that nearest neighbor's single class. The output in the KNN regression is the object's property value. That's the ideal number including its k values of the nearest neighbors.

KNN itself is a method of classifier where the mechanism is only locally approximated and all calculations are postponed until the analysis of the function. Because this algorithm tends to focus on classification distance, when the functionalities reflect distinct physiological units or arrive in significantly various scales, the normalization of training dataset can greatly enhance its accuracy.

Parameter selection

The appropriate choice of k relies on the information; generally, higher values of k decrease the effect of the noise on the classification, but make boundaries less distinct between groups. By various heuristic methods, a strong k can be extracted (see hyperparameter optimization). The nearest neighbor algorithm is called the necessary evil in which the classification is expected being the class of its closest training set (i.e., while k = 1).

By the existence of noisy or insignificant features, the accuracy of the k-NN algorithm can be severely degraded, or if the feature scales are not compatible with their significance. To

enhance classification, significant study effort has been made towards filtering or balancing features. Through use of evolutionary algorithms towards enhance feature scaling is an especially popular [citation required] approach. Another traditional strategy is just to optimize features with training classes through the shared knowledge of it's data sets.

It is beneficial to select k to be an odd number in binary (two class) classification problems, as this prevents tied votes. In this setting, one common way to choose the experimentally optimized k is through the bootstrap process.

The Mathematical Concepts Behind KNN

KNN functions because of the deeply ingrained mathematical ideas it uses, just like almost anything else. The first step is to translate data points into feature vectors, or their mathematical significance, while implementing KNN. By finding the distance between the mathematical values of these points, the algorithm then operates. The Euclidean distance is the most common way of finding this distance, as shown below.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \qquad 2.12$$

In order to compute the distance between each data point and the test data, KNN runs this formula. The likelihood of these points is then found to be identical to the test data and graded based on which points share the greatest probabilities.

2.4 Logistic Regression

Logistic Regression is a Supervised Learning algorithm which is not a regression but a classification algorithm. It's being used to predict discrete values associated with a given set of independent variable values (binary values such as 0,1 or yes, no or true, false). In simple words, through fitting the data towards a logit function, it predicts the likelihood of occurrence of the event. It is, thus, often referred to as logit regression. Since the likelihood is estimated, its performance values are between 0 and 1 because.

Suppose, we've got a puzzle to solve. There are only two outcome scenarios, either you or you don't solve it. Now imagine, in an effort to understand which topics we are strong at, when we are given a wide variety of puzzles or quizzes. If we have been given a trigonometry-based tenth grade dilemma, we are 70 percent likely to solve that. The result of this research will be something like this. But from the other hand, the chance of receiving an answer is only 30 percent if it is a fifth grade history question. Logistic Regression gives us this.

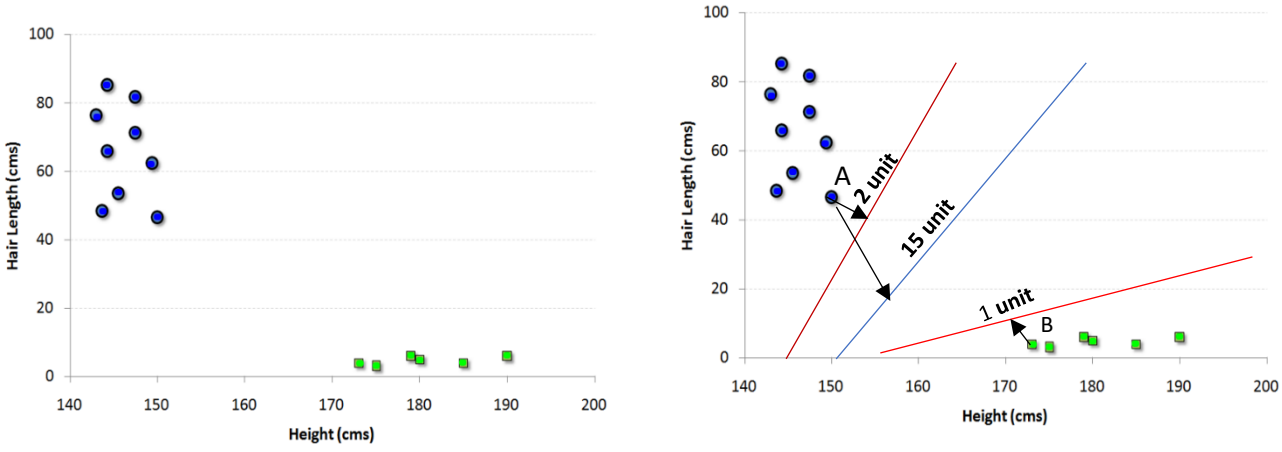
$$logit(p) = ln\left(\frac{p}{(1-p)}\right) \quad 2.13$$

In equation 2.13, p is the probability that there will be a characteristic of interest. Choose parameters which maximize the probability of observing the values of the sample rather than just minimize the sum of squared errors as in normal regression.

2.5 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a method of classification. and in this algorithm, we plot then each data element as a point in n-dimensional space at which n represents the number of features which we have, with its value of each function being its valuation of the specific coordinate

For example, if we had only two features like the Height and the Hair length of a person, we would then plot these two variables through two dimensional spaces that each point has two coordinates These coordinates are considered as support vectors.



(a) (b)

Figure 2. 9 SVM Explanation

Now, we're going to find a line that divides the data between the two different groups of data, which is shown in figure 2.8(a). This will be the line in such a way that the distances from the nearest point in each of the two groups will be the most distant.

In the example shown figure 2.8(b), the line that divides the data into two distinctly classified groups is the black line, since the two closest points are the most distant from the line. This is our ranking line. Finally, based on how far the test data lands from either side of the line that's what category the new data can be classified as.

2.6 Naive Bayes

Naive Bayes is a linear classifier focused on Bayes theory, assuming independence between predictors. In simple terms, the Naive Bayes classification algorithm of a specific feature in a class is not correlated with the presence of any other function. For example, if the fruit is red, round while approximately 3 inches in radius it can be considered an apple. Even though this value depends on one another or on its existence of the other functionalities the naive Bayes classifier might consider that most of these characteristics would individually make a contribution to the likelihood that the fruit would be an apple.

The Naïve Bayesian model is easy to build and is particularly useful for very large data sets. In addition to simplicity, Naive Bayes is known to perform even highly sophisticated classification methods.

Bayes Theorem offers a way to calculate P(c|x) of P(c), P(x) and P(x|c) of the posterior probability.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \qquad 2.14$$

In this,

- P(c|x) is the posterior probability of the target class given the predictor attribute.
- P(c) is the class's prior probability.
- P(x|c) is the probability of a given class predictor.
- P(x) is the prior predictor probability.

2.7 Decision Tree

Decision Tree is one of the most popular algorithms in the community and has been used quite quite often. It is a method of supervised learning algorithm which is most commonly used for classification problems. Surprisingly, this works with both categorical as well as continuous dependent variables. In this algorithm, we have divided the population into two or more homogeneous sets. This is framed in terms of the most significant attributes/independent variables to make as distinct groups as possible.

In the figure 2.9, we can see that the population is split into four separate categories consisting of multiple attributes to define 'whether or not they will play.' It uses a variety of techniques such as Gini, Information Gain, Chi-square, entropy to divide the population into various heterogeneous categories.

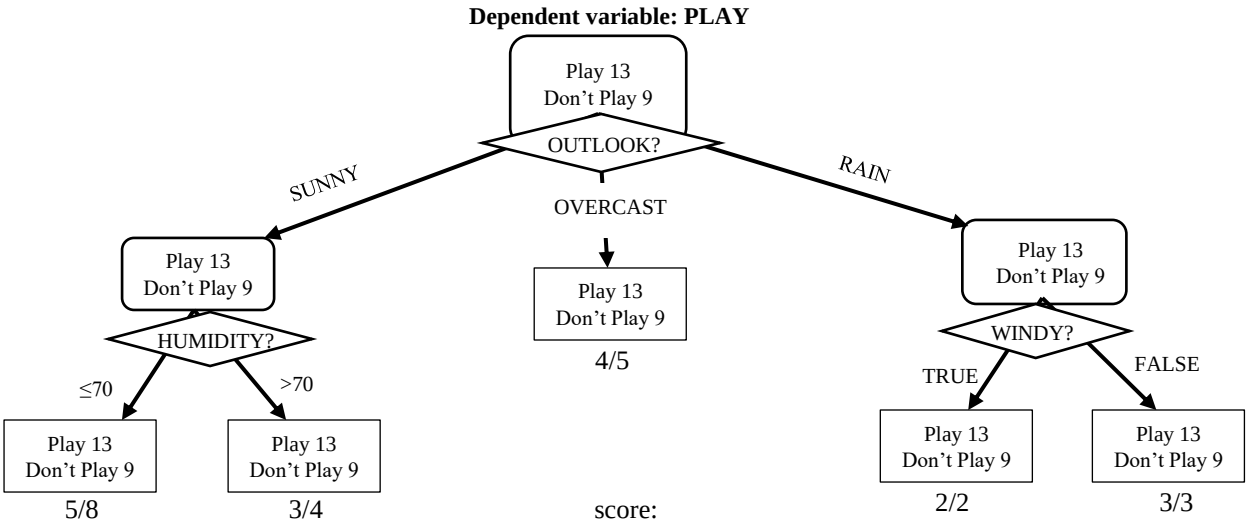


Figure 2. 10 Decision Tree Explanation

The easiest way of understanding how the decision tree operates is to play Jezzball – a classic Microsoft game (figure: 2.10). Supposedly, we have an area with moving walls, and we need to create walls that allow the maximum area to be cleared out of the balls.

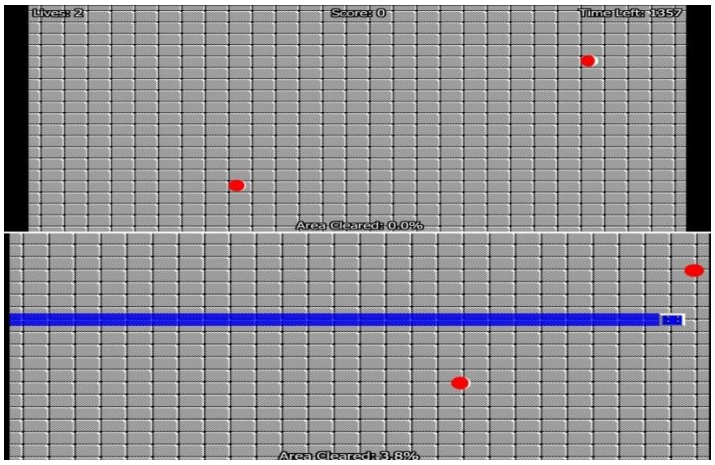


Figure 2. 11 Jezzball- a classic Microsoft game

So, every time we split each room with a wall, we're trying to create two different populations in the same room. Decision trees work in a very similar way by dividing the population into as many groups as possible.

2.8 Neural Network Supervised (MLPClassifier)

Artificial Neural Networks, or ANNs for short, are commonly used in many applications today, including classification, and there are many libraries and frameworks dedicated to quickly creating Neural Networks. When compared to a simple Scikit-Learn library, most of these frameworks and tools take several lines of code to implement.

The MLPClassifier from Scikit-Learn is one of the easiest to use Neural Networks for classification.

MLPClassifier stands for Multi-layer Perceptron Classifier, which is connected to a Neural Network by its name. Unlike other classification algorithms like Support Vectors or Naive Bayes Classifier, MLPClassifier is based on a mathematical model.
However, MLPClassifier is similar to Scikit-Learn's other classification algorithms in that it needs no more effort to implement than Support Vectors, Naive Bayes, or any other Scikit-Learn classifier.

The Multi-layer Perceptron (MLP) itself is a supervised machine learning algorithm which trains on a dataset to learn a function $f(.): R^m \rightarrow R^0$, where represents the number of input dimensions and is the amount of output dimensions. This could learn a non-linear function approximator for classification or regression given a set of characteristics $X = x_1, x_2, \ldots., x_m$ and a target y. It differs from logistic regression in that one or much more non-linear layer upon layer, known as hidden layers, may exist between both the input and output layers. Figure 2.11 shows a scalar output MLP with one hidden layer.

The input layer, on the left, is made up of a group of neurons called $\{x_i | x_1, x_2, \ldots., x_m\}$ that represent the input features. The variables from the previous layer are transformed within each neurons in the hidden layer using a weighted linear summation $\omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_m x_m$accompanied by a non-linear activation

function $g(.): R \rightarrow R$ - similar to the hyperbolic tan function. The values from the last hidden layer are passed to the output layer, which converts them to output values.

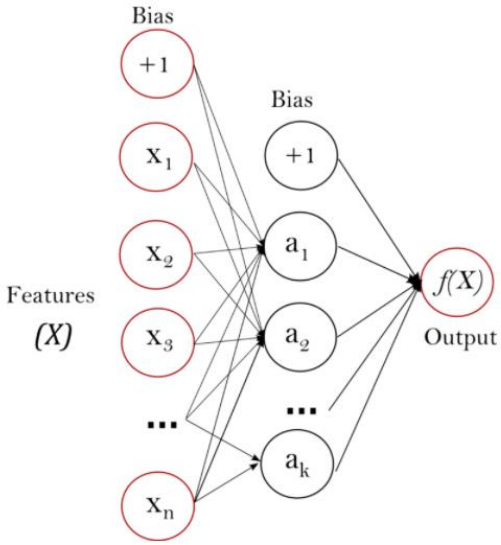


Figure 2. 12 One of The Multi-layer Perceptron (MLP) hidden layer.

The public attributes coefs_ and intercepts_ are contained within the module. The weight matrices at index represent the weights among layer i as well as layer i+1, and coefs_ is a collection of them. The vector at index reflects the bias values, while intercepts_ is a collection of bias vectors.

The public attributes coefs_ and intercepts_ are contained within the module. The weight matrices at index represent the weights among both layer i as well as layer i+1, while coefs_ is

a collection of them. The bias vector at index reflects the bias values applied to layer i+1, and intercepts_ is a collection of bias vectors.

2.9 Ridge Classification

RidgeClassifier is a classifier version of the Ridge regressor. This classifier converts binary goals to -1, 1 before treating the problem as a regression problem and optimizing the same goal as before. The predicted class is determined by the regressor's prediction symbol. The problem is viewed as multi-output regression for multiclass classification, and the expected class corresponds to the output with the highest value.

A (penalized) lower square loss could seem questionable in place of the more traditional logistical or hook losses, in place of a model classification. All these models can however direct to closely related cross-validation values in terms of accuracy or accuracy/recall in practice, while the lowest penalized losses used in the Ridge Classifier make it possible to choose numerical solvers of a very different computational performance.

The RidgeClassifier can be considerably faster than LogisticRegression for example including a wide range of positions, because the projection matrix $(X^T X)^{-1} X^T$ can only be computed once.

2.10 AdaBoost Classifier

Ada-boosting or adaptive boosting is one of Yoav Freund and Robert Schapire's ensemble boosting classification of 1996. It combines multiple classificators to improve classification accuracy. An iterative ensemble approach is AdaBoost. By combining several poor performance classifier so that you get a strong classifier with high accuracy, AdaBoost classifier creates a strong classifier. The general concept behind Adaboost here is to set the weights of the classifiers and to help train the sample data within every iteration so that the unusual observations are predicted accurately.

Any machine learning algorithm that accepts weights on the training set can be used as a base classifier. Adaboost must fulfill two requirements:

1. The classifier should be interactively trained using a variety of weighted training examples.
2. It tries to provide an excellent match for these examples in each iteration by minimizing training error.

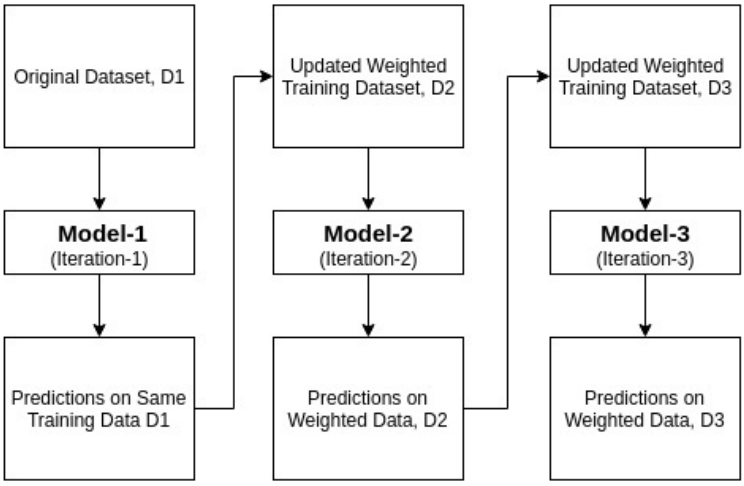


Figure 2. 13 The AdaBoost algorithm's operation

The AdaBoost algorithm's operation

1. Adaboost selects a training subset at random at first.
2. It trains the AdaBoost machine learning model iteratively by choosing the training set based on the last training's accurate prediction.
3. It gives incorrectly categorized observations a higher weight so that they have a higher chance of being classified in the next iteration.
4. It also assigns weight to the trained classifier in each iteration based on the classifier's accuracy. The more precise classifier will be given more weight.
5. This process runs until the complete training data is correct or the maximum number of estimators is reached without error.
6. To classify, take a vote in all the learning algorithms that you have developed.

2.11 Confusion Matrix

Calculation of the performance of a trained system is one of the measures that can be used to determine whether the model is reliable. It is therefore important to choose a tool that allows us to visualize this performance. Confusion matrix encapsulates the classification performance of the classifier with regards with some of the test data.

		Predicted	
		Negative	Positive
Actual	Negative	a	b
	Positive	c	d

Figure 2. 14 Confusion Matrix

In the context of the case study, the records throughout the confusion matrix have the following meaning:

- A is the number of correct predictions that an instance is negative;
- b is the range of incorrect statements that an instance is positive;
- c is the number of erroneous assumptions that an instance is negative,

- d is the proportion of true predictions that the case is positive.

New terms are derived from the confusion matrix:
Accuracy (AC) is the proportion of total number of assumptions that have been correct. This is defined by the equation

$$AC = \frac{a + d}{a + d + c + b} \quad 2.15$$

Recall or true positive rate (TP) is the ratio of positive samples which have occurred Identified correctly:

$$TP = \frac{d}{c + d} \quad 2.16$$

The false positive rate (FP) is the ratio of negative cases falsely predicted positive. It is defined by the following:

$$FP = \frac{b}{a + b} \quad 2.17$$

The true negative rate (TN) is expressed as the ratio of negative cases, have been correctly classified as measured by using equation:

$$TN = \frac{a}{a + b} \quad 2.18$$

The false negative rate (FN) is the ratio of positive cases incorrectly classified as negative as calculated using the equation:

$$FN = \frac{c}{c + d} \quad 2.19$$

Precision (P) is the ratio of positive cases predicted that were correct. This term is defined by the following equation:

$$P = \frac{d}{b + d} \quad 2.20$$

F1-score standard is the harmonic mean of precision and recall. The perfect model has an F-score of one.

$$F1 = \frac{2a}{2a + b + c} \quad 2.21$$

Matthews Correlation Coefficient (MCC) has a range of-1 to 1 where-1 indicates a factually false binary classifier, while 1 implies a totally correct binary classifier.

$$MCC = \frac{a \times d - b \times c}{\sqrt{(a+b)(a+c)(d+b)(d+c)}} \quad 2.22$$

2.12 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a methodology designed to reduce the dimensionality of certain sets of data, increase the usability but then at the same time reduce the loss of information. This technique extracts features that specifically change inputs, eliminating variables that do not provide important information to the data. Identifying such additional variables, the main components, reduces the problem of the own value/eigenvector and the new variables are characterized by the data set at hand, not a priori, making the PCA an affective data management methodology. Principal Component Analysis (PCA) technique is one of the most famous non-supervised dimensional reduction techniques.

The aim of the technique is to find the PCA space, which represents the direction of the maximum variance of the data. The PCA technique finds a lower dimensional space or a PCA space (W) that is used to transform the space.

Data (X = x_1, x_7,…., x_N) from the higher dimensional space (R^M) to the lower dimensional space (R^k), where N represents the total number of samples or observations and x_i represents i^{th} sample, pattern or observation. All samples are of the same size ($x_i \epsilon R^M$). In other words, each sample is represented by M variables, i.e., each sample is represented as an M-dimensional space point.

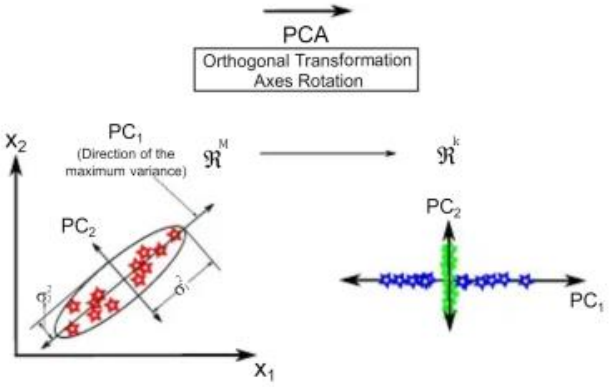


Figure 2. 15 Principal Component Analysis(PCA) explanation.

Figure 2.14 shows an example of two-dimensional data (x_1, x_2) where the original data is left with the original coordinates, i.e. x_1 and x_2, the variance of each variable is graphically represented and the direction of the maximum variance, i.e. PC_1, is shown; the original data is shown on the right. The main components are projected on the first (blue stars) and second (green stars).

2.13 Lime

LIME (Local Interpretable Model-agnostic Explanations) is an unique clarification approach which learns the interpretable model locally mostly around prediction to describe any classifier's prediction under an interpretable as well as faithful manner.
The influence of the LIME property on model interpretability

1. A model-independent, consistent explainer [LIME].
2. A framework for selecting a representative set that includes explanations [SP-LIME] to ensure that the model behaves consistently when replicating human reasoning. This representative collection will provide a global understanding of the model that is intuitive.Via feature engineer, LIME describes a prediction so that even non-experts can compare and build on an untrustworthy model.

LIME explains the forecast to enable even non-experts through feature engineering to compare and improve an unsustainable model. The following desirable properties should be included in an ideal model:

Interpretable

The qualitative understanding between the input variables and the answer should be provided. It should be understandable.

Local faithfulness

An explanation may not be completely faithful unless the model itself is described in its entirety. After saying that it should at least be faithful locally, it must replicate the behavior of the model next door to the predicted instance.

Agnostic Model

When giving examples, the explainer should be able to describe each model and should not make any assumptions about the model.

Perspective from afar

The explainer should give the consumer a representative collection to explain so that they have a general understanding of the model.

2.13.1 LIME assesses the following characteristics:

2.13.1.1 Data Representations That Can Be Interpreted

LIME uses a representation that humans can understand, regardless of the model's actual features. The term "interpretable representation" was coined to describe this. An interpretable representation will differ depending on the type of data we're working with, for instance:

1. In the case of text, it denotes the presence or absence of words.
2. In the case of an image, it denotes the presence or absence of superpixels (contiguous patch of similar pixels).
3. It is a weighted combination of columns for tabular results.

In short, even non-experts can understand LIME's explainer.

2.13.1.2. Interpretability-Fidelity a compromise

Locally replicate the model's actions and make it interpretable. LIME does this by minimizing the following:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}(f, g, \pi_x) + \Omega(g) \qquad 2.23$$

Variables in the equation

f: a unique predictor

x: exclusive features

g: an interpretation model, such as a linear model, a decision tree, or a collection of falling rule lists

Pi is a measure of how close an instance of z is to an instance of x in order to determine locality around x. It gives z' (perturbed instances) different weights depending on how far away they are from x.

First Term: the measure of g's unreliability in approximating f in the Pi-defined locality. In the original article, this is referred to as locality-aware failure.

Last term: a metric for the complexity of an explanation's model g. If the explanation model is a decision tree, for example, the width of the tree can be used, or in the case of linear explanation models, the number of non-zero weights can be used.

For future reference, here are some abbreviations.

1. ***x'*** (interpretable representation): This binary vector represents a human-readable version of the original model's actual features.
2. ***z'*** (perturbed sample): a percentage of x' elements that are not zero.
3. class mark ***f(z)***
4. ***g(z'):*** This is the model that LIME has mastered (explanation model).

Locality-aware loss is reduced while the second term is kept low enough to ensure both interpretability and local fidelity.

In order to ensure that both interpretability and local loyalty are minimized locality-conscious loss while maintaining the second term as low as possible for human interpretation. This is called ***Omega(g)*** for the remainder of the post.

LIME achieves local loyalty while optimizing locality-conscious loss.

2.13.1.3. Local Exploration By Sampling

To recap, g is the learning model, z' is an example of the training data, and f (z) is the y. Random uniform sampling from x' is used to construct a full training set. To put it another way, we make several z's out of a single row of x. (original training example).

These are then pi(x) weighted to concentrate on z', which is closer to x.

Equation 1 is optimized to learn the description model given this dataset and labels. To summarize, LIME's ability to provide explanations is not dependent on the form of original model (model agnostic).

2.13.1.4. Explanation in a Sparse Linear Format

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z \acute{\in} Z} \pi_x(z)\big(f(z) - g(\acute{z})\big)^2 \qquad 2.24$$

Assume that

1. ***g(z')*** = ***w. z'*** (Making the explanation model linear)
2. Loss that is locally conscious equals square loss

3. ***Pi(z) :*** *exp(-D(x,z)(2)/sigma(2))* (Samples are weighed based on their proximity)
4. Distance function ***D(x,z)***

Understanding the LIME algorithm

Algorithm 1 Sparse Linear Explanations using LIME

Require: Classifier f, Number of samples N
Require: Instance x, and its interpretable version x'
Require: Similarity kernel π_x, Length of explanation K
  $\mathcal{Z} \leftarrow \{\}$
  for $i \in \{1, 2, 3, ..., N\}$ **do**
    $z'_i \leftarrow sample\_around(x')$
    $\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$
  end for
  $w \leftarrow$ K-Lasso$(\mathcal{Z}, K)$ ▷ with z'_i as features, $f(z)$ as target
  return w

K places a limit on the number of variables that must be taken into account in the explanation. For example, K is the number of words to consider in a text, the number of superpixels in an image, and the number of columns in tabular data. If size(w) > K, we render Omega tend to infinity to achieve this. To summarize, LIME employs linear explainers to estimate the decision boundary of the original model.

2.13.1.5. SP-LIME is a submodular pick for illustrating templates.

LIME seeks to attribute human-understandable characteristics to a model's prediction. To do so, we must run the explanation model on a diverse but representative set of instances in order to generate a nonredundant explanation set that serves as a global representation of the model. Let's go through the prerequisites before we get into the algorithm:

1. ***B(Budget):*** The number of reasons the consumer is able to examine.
2. ***Select a Step:*** Selecting B instances from all instances is a difficult task.
3. ***W***(Explanation Matrix):***n***(number of samples)****d'***(human-understandable characteristics) matrix
4. ***I(j):*** In the description space, the value of component j is global.
5. ***V:*** Considered features for explanation
6. ***C (V, W, I):*** Calculates the total importance of all features in a set V that appear in at least one case.

$$c(V, W, I) = \sum_{j=1}^{d'} 1[\exists i \in V : W_{ij} > 0] I_j \quad (2.25)$$

Nonredundant converge intuition

$$Pick(W, I) = \underset{V, |V| \leq B}{argmax}\, c(V, W, I) \quad (2.26)$$

Steps in the Algorithm 2

Algorithm 2 Submodular pick (SP) algorithm

Require: Instances X, Budget B
  for all $x_i \in X$ **do**
    $\mathcal{W}_i \leftarrow$ **explain**(x_i, x_i') ▷ Using Algorithm 1
  end for
  for $j \in \{1 \dots d'\}$ **do**
    $I_j \leftarrow \sqrt{\sum_{i=1}^{n} |\mathcal{W}_{ij}|}$ ▷ Compute feature importances
  end for
  $V \leftarrow \{\}$
  while $|V| < B$ **do** ▷ Greedy optimization of Eq (4)
    $V \leftarrow V \cup \text{argmax}_i \, c(V \cup \{i\}, \mathcal{W}, I)$
  end while
  return V

1. Apply the interpretation model to all occurrences (all x's).
2. Calculate the relative value of individual components on a global scale.
3. Iteratively add the instance with the highest maximum coverage benefit to maximize the coverage function.
4. Go back to V. (representative nonredundant explanation set)

As can be shown, LIME possesses all four desirable properties of an ideal model explainer.: Considered features

Chapter 3
Data Overview

An overview of the database used in this project is provided in this chapter. This is crucial because to ensure effective strategic decision-making, we can view data and information needed.

3.1 Dataset of Credit Card Fraud

The dataset includes credit card purchases made by European cardholders in September 2013.We have used the latest third version of this dataset which is updated in march 2018.
This dataset presents two-day transactions, of which we have 492 frauds out of 284,807 transactions. The dataset is strongly unbalanced, accounting for 0.172 percent of all transactions in the positive class (fraud).

It only includes numerical input variables that are the product of a transformation of a PCA. Unfortunately, we do not include the original characteristics and further background details about the data due to confidentiality problems. Features V1,V2, ,...,V28 are the key components obtained with PCA, 'Time' and 'Number' are the only features not transformed with PCA.
The 'Time' feature includes the seconds in the dataset between each transaction and the first transaction. The 'Number' function is the Amount transaction, which can be used for example-dependent cost-sensitive learning. The answer variable is the 'Class' function, which takes value 1 in the case of fraud and 0 otherwise.

#	Time	V1	…	V28	Amount	Class
1	0	-1.359807134	…	-0.021053053	149.62	0
2	0	1.191857111	…	0.014724169	2.69	0
3	1	-1.358354062	…	-0.059751841	378.66	0
4	1	-0.966271712	…	0.061457629	123.5	0
5	2	-1.158233093	…	0.215153147	69.99	0
…	…	…	…	…	…	…
284803	172786	-11.88111789	…	0.823730961	0.77	0
284804	172787	-0.732788671	…	-0.053527389	24.79	0
284805	172788	1.91956501	…	-0.026560829	67.88	0
284806	172788	-0.24044005	…	0.104532821	10	0
284807	172792	-0.533412522	…	0.013648914	217	0

Table 3. 1 Dataset of Credit Card Fraud

3.1.1 Time Feature

Time indicates the seconds which have elapsed since the first transaction. Once the values are graphically represented (see Figure 3.1), it can be checked that transactions that have occurred over a span of two days are stored in the database. The data indicates bimodal activity in which there is a substantial decrease in the number of transactions after a period of approximately 24 hours. It is fair to assume that since they are night hours, this fall arose. Finally, this variable

is assumed to be omitted because it is not applicable to learning the model, because until the last transaction, the data is very similar to others.

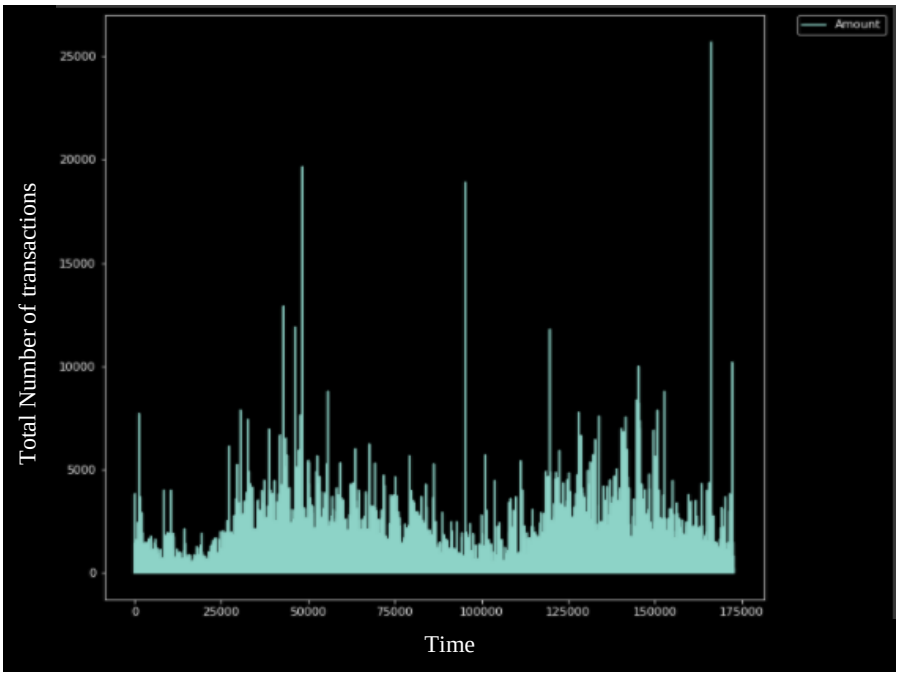


Figure 3. 1 Time Function Allocation of the Dataset

3.1.2 Amount Feature

The amount of cash in each transaction is a function called Amount. The highest transaction with this collection of data is $25,691.16, while the transaction average is $88.35. Figure 3.2 shows that the data is mainly clustered at very low values close to zero, while the maximum value found is approximated by just a few transactions. The representation of the amount of money for each transaction (see Figure 3.3) indicates, on the other hand, certain values which differ from each other. These are referred to as outliers because they are transactions in which a significant sum of money is exchanged in this situation. These principles logically draw the attention of potential fraud, but this is something that fraudsters want to avoid altogether. Existing evidence indicates that small sums of money have also been moved by fraudsters to continue stealing in an undetectable way.

3.1.3 Class Feature

Figure 3.4 represents the feature called Class, which gives information that if the transactions are fraudulent or not, this variable takes value 1 in case of fraud and 0 otherwise.

This feature shows that there is a minimum percentage of fraudulent cases which represent 0.17% of all data. While non-fraudulent cases equal 99.83%. It is concluded that the data is highly imbalanced, which requires choosing appropriate measures to divide the data and make the training of the system effective.

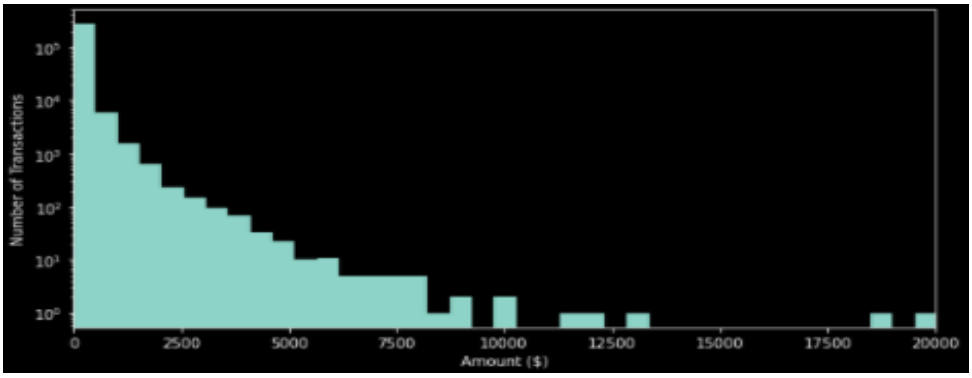


Figure 3. 2 Amount per transaction

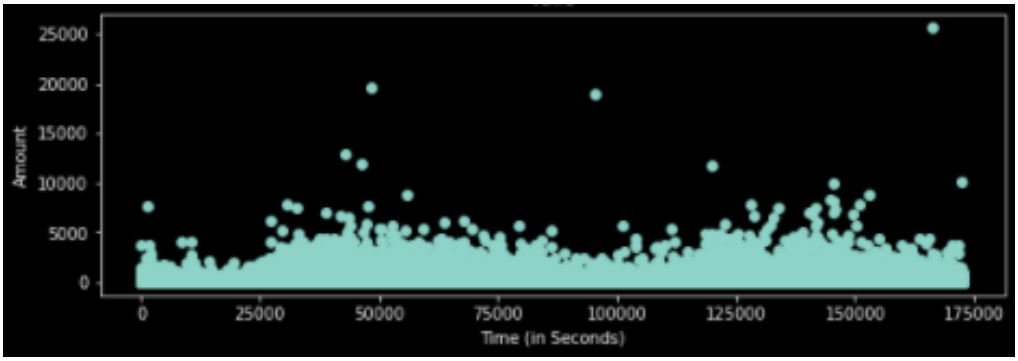


Figure 3. 3 Tome of transaction vs amount

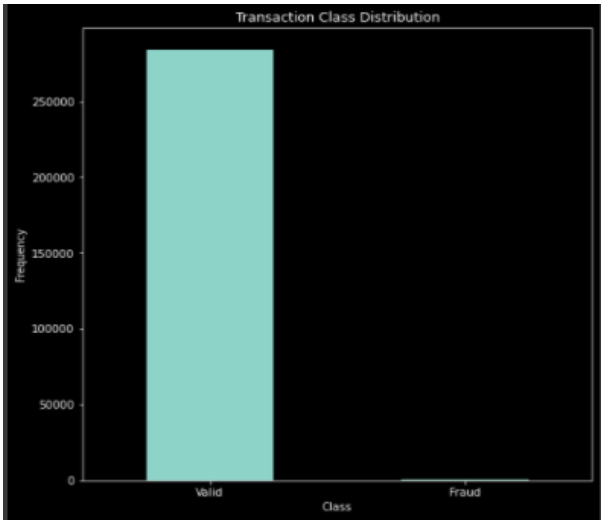


Figure 3. 4 Transaction class distribution

3.1.4 V-features

It is also beneficial to observe the V-feature histogram representation (see figure 3.5). This gives a basic concept of the distribution of data. It is also necessary to check whether there is a significant correlation between the characteristics, particularly with regard to the class function. A matrix of correlations between all features is shown in figure 3.6. This representation stresses that there are few class-related features and that while there are several features in the class,

There are very few important associations in the results. This implies that the characteristics are essentially Principal Components, the product of the previous PCA planning that the dataset had.

Finally, it can be seen that the features of Time1 and Amount do not correspond with the Class function, so they are not important in the learning phase of the system.

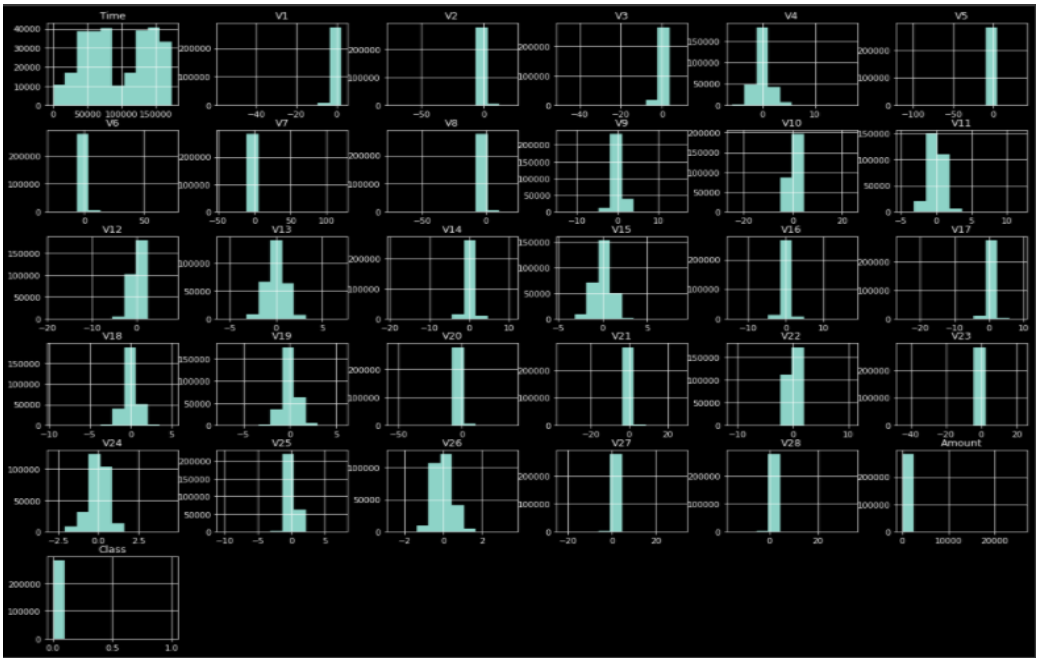


Figure 3. 5 Frequency for each Dataset Function

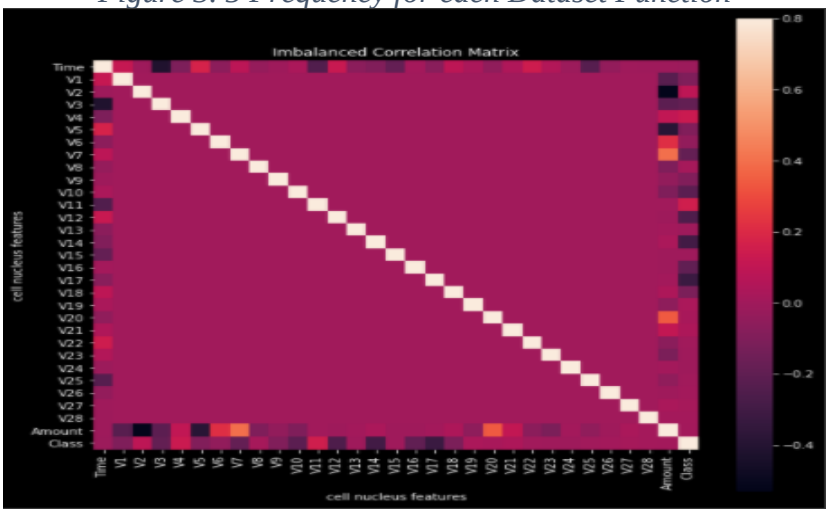


Figure 3. 6 The Dataset Correlation Matrix

Chapter 4
Data Sorting and Exploration

In this chapter, we have sort out the dataset and explore to prepare the data. We will be using python, TensorFlow(version 2.2) and colab notebook for implementation.

4.1 TensorFlow

TensorFlow is a quick numerical computing library that is open source.Google developed and maintains it, and it's open source under the Apache 2.0 license. While there is access to the underlying C++ API, the API is ostensibly for the Python programming language.

Unlike other numerical libraries for Deep Learning, such as Theano, TensorFlow was created with the aim of being used in both research and production systems, including Google's RankBrain and the fun DeepDream project.

It can run on single-processor computers, GPUs, mobile devices, and large-scale distributed systems involving hundreds of machines.

4.2 Colab

Colaboratory, or' Colab' in shorter, itself is a Google Research product. Colab enables anyone through the browser to compose as well as implement arbitrary python scripts, and is particularly applicable suitable for machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that needs no configuration to use, thus providing free access to GPUs, including device resources.

The open source project on which Colab is based is Jupyter. Colab allows to use Jupyter notebooks and share them with others without downloading, installing or running anything.

4.3 Importing Packages for Data Sorting and Exploration

Pandas to work with data, NumPy to work with arrays, scikit-learn for data split, building and evaluating classification models, and finally matplotlib for visualization will be the key packages to begin with.

```
# IMPORTING PACKAGES

import pandas as pd # data processing
import numpy as np # working with arrays
import matplotlib.pyplot as plt # visualization

from sklearn.preprocessing import StandardScaler # data normalization
from sklearn.model_selection import train_test_split # data split
```

Figure 4. 1 Importing packages for data sorting and exploration

4.4 Importing the data

In order to mount the environment with google drive, we need to import drive function from google.colab (figure 4.2). As we import the data using the ‘pd.read_csv’ method and print the

data. The dataset we have used is from Kaggle Credit Card Fraud Detection. The key components obtained by PCA are features V1 to V28. We will disregard the time function that is of no use in constructing the models. Further the existing features are "Time" feature containing transaction time, the "Amount" feature containing the total quantity of money being transacted as well as the "Class" feature containing about whether or not the transaction itself is a case of fraud. The details of it has showed in figure 4.3.

Input

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Output:

```
Mounted at /content/gdrive
```

Figure 4. 2 Mount the environment with google drive.

Input:

```
# IMPORTING DATA
data= pd.read_csv('/content/gdrive/Shareddrives/credit card Froud /dataset/Credit Card Fraud Detection 310_23498_bundle_archive/creditcard.csv')
print(data.head())
```

Output:

```
   Time        V1        V2        V3  ...       V27       V28  Amount  Class
0   0.0 -1.359807 -0.072781  2.536347  ...  0.133558 -0.021053  149.62      0
1   0.0  1.191857  0.266151  0.166480  ... -0.008983  0.014724    2.69      0
2   1.0 -1.358354 -1.340163  1.773209  ... -0.055353 -0.059752  378.66      0
3   1.0 -0.966272 -0.185226  1.792993  ...  0.062723  0.061458  123.50      0
4   2.0 -1.158233  0.877737  1.548718  ...  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

Figure 4. 3 Importing the data

4.5 Exploratory Data Analysis and Processing

In this process, we have analyzed the data and perspire the by splitting, training and testing. In addition to analysis of the noisy data, we have applied PCA and StandardScaler module from scikit-learn preprocessing import for standardization.

4.5.1 Cases Count

In the figure, 4.4 we can see that there are only 492 cases of fraud out of 284,807 samples, which is only 0.17 per cent of the total samples in the dataset. So, we can state that the data we deal with is highly imbalanced.

Input:

```
cases = len(data)
fraud = len(data[data.Class == 1])
valid = len(data[data.Class == 0])
fraud_percentage = (fraud)/float(valid)
print('Total number of cases are {}'.format(cases))
print('Fraud Cases: {}'.format(fraud))
print('Valid Transactions: {}'.format(valid))
print('Fraud Percentage: {}'.format(fraud_percentage))
```

Output:

```
Total number of cases are 284807
Fraud Cases: 492
Valid Transactions: 284315
Fraud Percentage: 0.0017304750013189597
```

Figure 4. 4 Determine the number of fraud and valid cases in datasets.

Using the 'describe' method, a statistical view of both fraud and valid transaction amount data has generated which is shown in figure 4.5 and 4.6

Input:

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

Output:

```
Amount details of the fraudulent transaction
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

Figure 4. 5 Statistical view of fraud transaction amount

Input:

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

Output:

```
Amount details of the valid transaction
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

Figure 4. 6 Statistical view of valid transaction amount

4.5.2 Splitting Data into X and Y Values

We have divided the information into input parameters and output value formats in this process, shown in figure 4.7. And in a training set and testing set, define the value as the independent (X) and the dependent variables (Y). Furthermore, we divided the information into a training set and a testing set using the specified variables.

4.5.3 Training and Testing The Splitting Data

For modelling and assessing, training and testing sets are used. We use the 'train test split' algorithm to split the data. And the samples shown in Figure 4.8 are written.

Input:

```
X = data.drop(['Class'], axis = 1)
Y = data["Class"]
print(X.shape)
print(Y.shape)

xData = X.values
yData = Y.values
```

Output:

```
(284807, 30)
(284807,)
```

Figure 4. 7 Splitting data into X and Y values

Input:

```
from sklearn.model_selection import train_test_split

xTrain, xTest, yTrain, yTest = train_test_split(
        xData, yData, test_size = 0.2, random_state = 42)
print('X_train samples : ', xTrain[:1])
print('X_test samples : ', xTest[0:1])
print('y_train samples : ', yTrain[0:20])
print('y_test samples : ', yTest[0:20])
```

Output:

```
xTrain samples :  [[ 1.43352000e+05  1.95504092e+00 -3.80782711e-01 -3.15012853e-01
   3.30155452e-01 -5.09374248e-01 -8.61974532e-02 -6.27977906e-01
   3.59937221e-02  1.05456030e+00 -3.04413407e-02  6.24995774e-01
   1.69149569e+00  1.25579040e+00 -2.53266461e-01 -3.31694981e-01
   3.07252346e-01 -9.30843692e-01  6.51665792e-01  1.67986640e-01
  -1.25389994e-01  2.38197424e-01  9.68304938e-01  5.32080152e-02
  -2.78601509e-01 -4.49993014e-02 -2.16780337e-01  4.51682478e-02
  -4.71447917e-02  9.99000000e+00]]
xTest samples :  [[ 4.15050000e+04 -1.65265066e+01  8.58497180e+00 -1.86498532e+01
   9.50559352e+00 -1.37938185e+01 -2.83240430e+00 -1.67016943e+01
   7.51734390e+00 -8.50705864e+00 -1.41101844e+01  5.29923635e+00
  -1.08340065e+01  1.67112025e+00 -9.37385858e+00  3.60805642e-01
  -9.89924654e+00 -1.92362924e+01 -8.39855199e+00  3.10173537e+00
  -1.51492344e+00  1.19073869e+00 -1.12767001e+00 -2.35857877e+00
   6.73461329e-01 -1.41369967e+00 -4.62762361e-01 -2.01857525e+00
  -1.04280417e+00  3.64190000e+02]]
yTrain samples :  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
yTest samples :  [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Figure 4. 8 Training and testing the splitting data

4.5.4 Analyze the noisy test and Train Data

From Figure 4.9, we have pointed out how many noisy training and test sets are by applying PCA.

Input:

```
from sklearn.decomposition import PCA

pca = PCA()
xTrain = pca.fit_transform(xTrain)
xTest = pca.transform(xTest)
xTrain_transformed = pca.transform(xTrain)


fig, (ax) = plt.subplots(ncols=1, figsize=(10, 7))

for l, c, m in zip(range(0,2), ('blue', 'red'), ('^', 's')):
    ax.scatter(xTrain_transformed[yTrain == l, 0],
               xTrain_transformed[yTrain == l, 1],
               color=c,
               label='class %s' % l,
               alpha=0.5,
               marker=m
               )

ax.set_title('Training dataset after PCA')
ax.set_xlabel('1st principal component')
ax.set_ylabel('2nd principal component')
ax.legend(loc='upper right')
ax.grid()

plt.tight_layout()

plt.show()
```

Output:

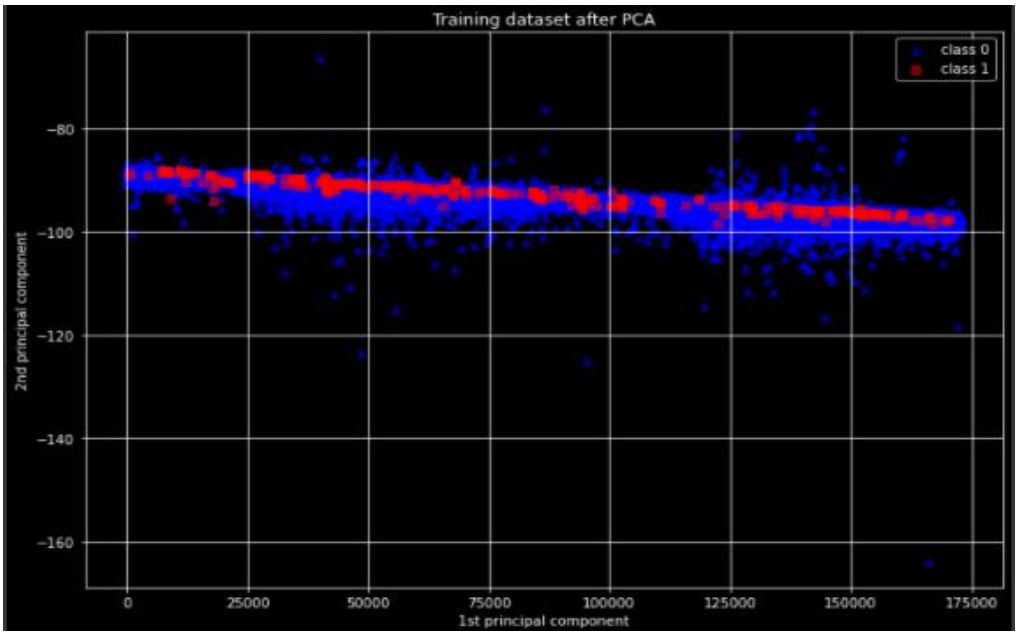


Figure 4. 9 Training dataset after PCA Applied

4.5.5 Standardize the noisy test and Train Data

To minimize and standardize noisy data by applying StandardScaler from sci-kit-learn preprocessing. In addition to visualizing the PCA standardized dataset conducted in Figure 4.10

Input:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
xTrain = sc.fit_transform(xTrain)
xTest = sc.transform(xTest)
xTrain_std_transformed = pca.transform(sc.transform(xTrain))
explained_variance = pca.explained_variance_ratio_

fig, (ax) = plt.subplots(ncols=1, figsize=(10, 7))

for l, c, m in zip(range(0,2), ('blue', 'red'), ('^', 's')):
    ax.scatter(xTrain_std_transformed[yTrain == l, 0],
               xTrain_std_transformed[yTrain == l, 1],
               color=c,
               label='class %s' % l,
               alpha=0.5,
               marker=m
               )

ax.set_title('Standardized training dataset after PCA')
ax.set_xlabel('1st principal component')
ax.set_ylabel('2nd principal component')
ax.legend(loc='upper right')
ax.grid()

plt.tight_layout()

plt.show()
```

Output:

Figure 4. 10 Standardized dataset of training after PCA

Chapter 5
Modeling

Artificial Neural Network (ANN) and none different types of classification models have been developed in this section, namely Decision Tree, K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machine (SVM), Random Forest, Neural Network Supervised (MLPClassifier), Ridge Classification, AdaBoost Classification and Naive Bayes. These are the most common models used to solve classification problems, although there are several more models that we can use. All of these models have been developed using the algorithms given by the sci-kit-learn package. We only used the Keras for the ANN model.

5.1 Artificial Neural Network (ANN)

In order to build an Artificial Neural Network model, we have developed and compiled the model with keras package.

5.1.1 Develop Keras Model

We used "sequential" models to construct the neural network. This top-down approach helps to create and play with the form and layers of the Neural net architecture. The first layer will have the amount of characteristics that can be corrected using "input dim" In this situation, we set it at 40.

It is not a very simple process to build Neural Networks. Before a successful model is constructed, there are many trials and failures that take place. Using the "Dense" class in Keras, we have created a Completely Connected network structure. The Neuron is the first statement that the dense layer has to give.

Using the activation statement, the activation function can be set. In this case, we have used the 'Rectified Linear' Unit as the activation function. Other options such as "Sigmoid" or "TanH" are available, but "RELU" is a more general option and a better one.

5.1.2 Compiling The Model

The next step after model specification is compiling the model. For model compilation, TensorFlow is used. Compilation is the phase in which parameters for model training and forecasts are set. In the context, CPU/GPU or distributed memories can be used.

We have defined a loss function which is used for the various layers to calculate weights. The optimizer changes the rate of learning and goes through different weight sets. We used "Binary Cross Entropy" as the loss function in this instance. We used "ADAM" in the case of the optimizer, which is an effective stochastic gradient descent algorithm.

It is used very widely for tuning. Finally, since it is a classification question, the classification accuracy, accuracy, recall and Matthews correlation coefficient(MCC) identified by the metrics argument will be collected and published. For MCC, we have the principles summarized.
The process of building the ANN model has shown in figure 5.1 and 5.2
Input:

```
# building our Neural Network

classifier = Sequential()#empty model
classifier.add(Dense(40 , input_dim = 30 , activation = 'relu'))
classifier.add(Dense(30 , input_dim = 40 , activation = 'relu'))
classifier.add(Dense(20 , input_dim = 30 , activation = 'relu'))
classifier.add(Dense(10 , input_dim = 20 , activation = 'relu'))
classifier.add(Dense(6 , input_dim = 10 , activation = 'relu'))
classifier.add(Dense(4 , input_dim = 6 , activation = 'relu'))
classifier.add(Dense(1, input_dim = 4 , activation = 'sigmoid'))
classifier.summary()
```

Output:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 40)                1240
_________________________________________________________________
dense_1 (Dense)              (None, 30)                1230
_________________________________________________________________
dense_2 (Dense)              (None, 20)                620
_________________________________________________________________
dense_3 (Dense)              (None, 10)                210
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 66
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 28
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 5
=================================================================
Total params: 3,399
Trainable params: 3,399
Non-trainable params: 0
_________________________________________________________________
```

Figure 5. 1 Developing sequential model (ANN model)

```
from keras import metrics
metrics = [
    keras.metrics.Precision(name="precision"),
    keras.metrics.Recall(name="recall"),
    keras.metrics.Accuracy(name="Accuracy")
]

from keras import backend as K
def matthews_correlation_coefficient(y_true, y_pred):
  tp = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
  tn = K.sum(K.round(K.clip((1 - y_true) * (1 - y_pred), 0, 1)))
  fp = K.sum(K.round(K.clip((1 - y_true) * y_pred, 0, 1)))
  fn = K.sum(K.round(K.clip(y_true * (1 - y_pred), 0, 1)))

  num = tp * tn - fp * fn
  den = (tp + fp) * (tp + fn) * (tn + fp) * (tn + fn)
  return num / K.sqrt(den + K.epsilon())


def f1(y_true, y_pred):
  tp = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
  predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
  precision = tp / (predicted_positives + K.epsilon())
  possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
  recall = tp / (possible_positives + K.epsilon())
  return 2*((precision*recall)/(precision+recall+K.epsilon()))

classifier.compile(loss = 'binary_crossentropy' , optimizer = 'adam' , metrics = [metrics, matthews_correlation_coefficient,f1]  )
```

Figure 5. 2 Compiling sequential model (ANN Model)

5.2 Decision Tree

To construct the model, we used the 'DecisionTreeClassifier' algorithm. We have listed the 'max depth' within the algorithm to be '4' which implies that we allow the tree to split four times and the 'criterion' to be 'entropy' which is most similar to the 'max depth' but specifies when the tree should stop splitting. Finally, in the 'tree-Pred' variable, we have installed and stored the expected values. Shown in figure 5.3

```
# Decision Tree
from sklearn.tree import DecisionTreeClassifier # Decision tree algorithm
tree_model = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
tree_model.fit(xTrain, yTrain)
tree_Pred = tree_model.predict(xTest)
```

Figure 5. 3 Construction of Decision Tree model

5.3 K-Nearest Neighbors (KNN)

Using the 'KNeighborsClassifier' algorithm, we developed the model and mentioned the 'n_neighbors' as '5'.The value of the 'n_neighbors' is chosen at random, but it can be selected optimistically by iterating the set of outcomes, followed by fitting and storing values that have predicted into the 'knn_Pred' vector. Figure 5.4 shows the Construction of K-Nearest Neighbors (KNN) model.

```
# 2. K-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier # KNN algorithm
n = 5

knn = KNeighborsClassifier(n_neighbors = n)
knn.fit(xTrain, yTrain)
knn_pred = knn.predict(xTest)
```

Figure 5. 4 Construction of K-Nearest Neighbors (KNN) model

5.4 Logistic regression

For logistic regression, by using the 'LogisticRegression' algorithm, we kept the model in a more simplified way and, as usual, fitted and stored the predicted variables in the 'lr_red' variable. Construction of Logistic regression model has shown in figure 5.5

```
# 3. Logistic Regression
from sklearn.linear_model import LogisticRegression # Logistic regression algorithm
lr = LogisticRegression()
lr.fit(xTrain, yTrain)
lr_pred = lr.predict(xTest)
```

Figure 5. 5 Construction of Logistic regression model

5.5 Support Vector Machine (SVM)

Using the 'SVC' algorithm, we designed the Support Vector Machine model and we did not specify anything inside the classifier as we planned to use the 'rbf' kernel as the default kernel. After that, forward to fitting the model, we preserved the predicted values in 'svm Pred'. Support Vector Machine (SVM) model Construction has shown in figure 5.6.

```
# 4. SVM
from sklearn.svm import SVC # SVM algorithm
svm = SVC()
svm.fit(xTrain, yTrain)
svm_pred = svm.predict(xTest)
```

Figure 5. 6 Support Vector Machine (SVM) model construction.

5.6 Random forest

We developed a random forest model using the 'RandomForestClassifier' algorithm and specified the 'max_depth' to be 4, much like how we built the model of the decision tree. Finally, the values are adapted and stored in the 'rfc_Pred '. Notice that the key difference between the decision tree and the random forest is that, while the random forest uses randomly selected features to create different models, the decision tree uses the entire dataset to construct a single model. That's why the random forest model versus a decision tree is used. In figure 5.7, construction has demonstrated.

```
# Building the Random Forest Classifier (RANDOM FOREST)
from sklearn.ensemble import RandomForestClassifier
# random forest model creation
rfc = RandomForestClassifier()
rfc.fit(xTrain, yTrain)
# predictions
rfc_pred = rfc.predict(xTest)
```

Figure 5. 7 Random Forest model construction

5.7 Naive Bayes

For naïve bayes model we have used GaussianNB, which implies for classification. We have fitted and stored the final prediction by the model in “NBC_pred”,which is shown in figure 5.8.

```
# 7.Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB
NBC = GaussianNB()
NBC.fit(xTrain, yTrain)
NBC_pred = NBC.predict(xTest)
```

Figure 5. 8 Construction of Naïve Bayes classification model.

5.8 Neural Network Supervised (MLPClassifier)

We built a Neural Network Supervised model using the 'MLPClassifier' algorithm. We've also introduced parameters solver, alpha, hidden_layer_sizes, and random_state. After that,

the expected values are fitted and stored in the 'nns_pred' vector. The construction of a Neural Network Supervised model is shown in Figure 5.9.

```
from sklearn.neural_network import MLPClassifier
nns = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=1)
nns.fit(xTrain,yTrain)
nns_pred = nns.predict(xTest)
```

Figure 5. 9 Construction of Neural Network Supervised (MLPClassifier) model.

5.9 Ridge Classification

RidgeClassifier was used to build Ridge Classification models, as the name suggests. The model's final prediction has been fitted and stored in "RC_pred," as shown in figure. 5.10

```
from sklearn.linear_model import RidgeClassifier
RC = RidgeClassifier()
RC.fit(xTrain, yTrain)
RC_pred = RC.predict(xTest)
```

Figure 5. 10 Construction of Ridge Classification model.

5.10 AdaBoost Classification

The 'AdaBoost Classification' algorithm is used to build an AdaBoost Classification model. N estimators parameter has also been introduced. Then you fit and store the expected values in the 'AdBC_pred' vector. Figure 5.11 illustrates the structure of an AdaBoost classification model.

```
from sklearn.ensemble import AdaBoostClassifier
AdBC = AdaBoostClassifier(n_estimators=100)
AdBC.fit(xTrain, yTrain)
AdBC_pred = AdBC.predict(xTest)
```

Figure 5. 11 Construction of AdaBoost Classification model.

Chapter 6
Model Evaluation

We have evaluated our constructed models in this chapter using the evaluation metrics given by the scikit-learn package. But we used the ANN model metrics function from the keras package. The accuracy, f1, recall, precision, Matthew's correlation coefficient (MCC) and finally the confusion matrix are the assessment metrics we have used. Additionally, we have used lime for model interpretation.

6.1 ANN Model Evaluation

To evaluate the ANN model, we have used evaluate method form keras package. (figure 6.1). AS classification metrics can't handle a mix of binary and continuous targets.

Input:

```
# Getting the results of ANN
classifier.evaluate(xTest,yTest)
```

Output:

```
1781/1781 [==============================] - 2s 1ms/step - loss: 0.0090 - precision: 0.8539 - recall: 0.7755 - Accuracy: 0.0175 - matthews_correlation_coefficient: 0.0427 - f1: 0.0427
[0.008979643695056438,
 0.8539325594902039,
 0.7755101919174194,
 0.017450230196118355,
 0.04267265647649765,
 0.04267265275120735]
```

Figure 6. 1 ANN Model Evaluation.

6.2 Recall

The fraction of the related records that are effectively recovered is a recall.

$$recall = \frac{|(relevent\ documents) \cap (relevent\ documents)|}{(relevent\ documents)}$$

For example, recall is the number of correct results, divided by the number of results that should have been returned, for a text search on a collection of documents.

Remembering is called sensitivity in binary classification. It can be interpreted as a probability that the query will retrieve a suitable document.

It is negligible to maintain recall of 100 percent by returning all documents in response to any question. Thus the, recall is not always enough, however the number of non-relevant records must also be calculated, for example, by measuring the accuracy as well. For measuring recall, we have used recall metrics from keras for our ANN model (fig 5.2)and for our six classifier model "recall_score" method from the scikit-learn package(6.2) to do it in python.

Input:

```
from sklearn.metrics import recall_score
rfc_recall = recall_score(yTest, rfc_pred)
print("The recall score  of Random Forest Classifier {}".format(rfc_recall))
tree_recall = recall_score(yTest, tree_Pred )
print("The recall score of Decision Tree {}".format(tree_recall))
knn_recall = recall_score(yTest, knn_pred )
print("The recall score of knn {}".format(knn_recall))
lr_recall= recall_score(yTest, lr_pred )
print("The recall score of Logistic Regression {}".format(lr_recall))
svm_recall = recall_score(yTest, svm_pred)
print("The recall score of svm {}".format(svm_recall))
NBC_recall = recall_score(yTest, NBC_pred)
print("The recall score of  Naive Bayes classifier {}".format(NBC_recall))
nns_recall = recall_score(yTest, nns_pred )
print("The recall score is Neural Network Supervised (MLPClassifier){}".format(nns_recall))
RC_recall = recall_score(yTest, RC_pred )
print("The recall score is Ridge Classification {}".format(RC_recall))
AdBC_recall = recall_score(yTest, AdBC_pred )
print("The recall score is AdaBoost  {}".format(AdBC_recall))
```

Output:

```
The recall score  of Random Forest Classifier 0.7755102040816326
The recall score of Decision Tree 0.7755102040816326
The recall score of knn 0.7857142857142857
The recall score of Logistic Regression 0.5816326530612245
The recall score of svm 0.6122448979591837
The recall score of  Naive Bayes classifier 0.8163265306122449
The recall score is Neural Network Supervised (MLPClassifier)0.8061224489795918
The recall score is Ridge Classification 0.4387755102040816
The recall score is AdaBoost  0.7653061224489796
```

Figure 6. 2 Recall Score of six different classification models by the scikit-learn package.

After reviewing the chapter 6.1 and 6.2, found recall scores have listed in Table 6.1.

#	Model	Recall score
1	Artificial Neural Network (ANN)	0.7755101919174194
2	Random Forest	0.7755102040816326
3	K-Nearest Neighbors (KNN)	0.7857142857142857
4	Logistic Regression	0.5816326530612245
5	Support Vector Machine (SVM)	0.6122448979591837
6	Naive Bayes Classifier	0.8163265306122449
7	Decision Tree	0.7755102040816326
8	Neural Network Supervised (MLPClassifier)	0.8061224489795918
9	Ridge Classification	0.4387755102040816
10	AdaBoost Classification	0.7653061224489796

Table 6. 1 Recall score of our models

6.3 Precision

The accuracy is the fraction of documents obtained that are important to the query:

$$precision = \frac{|(relevent\ documents) \cap (relevent\ documents)|}{(relevent\ documents)}$$

For instance, accuracy is the number of correct results, divided by the number of all returned results, for a text search on a collection of documents.

Precision takes into account all the documents obtained, but it can also be assessed at a specified cut-off rank, taking into account only the system's highest return performance. At n or P/n, this calculation is called precision. Precision is used for recall, the percent of all related documents that is returned by the scan. In order to provide a single calculation for a system, the two measurements are often used together in the F1 Score (or f-measure). For our six classifier model " precision_score" framework from the scikit-learn package(fig 6.3)and for ANN model (fig 5.2) we used precision metrics from keras for calculating precision to do it in python.

Input:

```
from sklearn.metrics import precision_score
rfc_prec = precision_score(yTest, rfc_pred)
print("The precision score of Random Forest Classifier {}".format(rfc_prec))
tree_prec = precision_score(yTest, tree_Pred )
print("The precision score of Decision Tree {}".format(tree_prec))
knn_prec = precision_score(yTest, knn_pred )
print("The precision score of knn {}".format(knn_prec))
lr_prec= precision_score(yTest, lr_pred )
print("The precision score of Logistic Regression {}".format(lr_prec))
svm_prec = precision_score(yTest, svm_pred)
print("The precision score of svm {}".format(svm_prec))
NBC_prec = precision_score(yTest, NBC_pred)
print("The precision score of  Naive Bayes classifier {}".format(NBC_prec))
```

Output:

```
The precision score of Random Forest Classifier 0.9868421052631579
The precision score of Decision Tree 0.8636363636363636
The precision score of knn 0.9390243902439024
The precision score of Logistic Regression 0.8636363636363636
The precision score of svm 0.967741935483871
The precision score of  Naive Bayes classifier 0.05956813104988831
```

Figure 6. 3 precision score of six different classification models by the scikit-learn package.

After analyzing chapter 6.1 and 6.3, the precision scores contained in Table 6.2

#	Model	Precision score
1	Artificial Neural Network (ANN)	0.8539325594902039
2	Random Forest	0.987012987012987
3	K-Nearest Neighbors (KNN)	0.9390243902439024
4	Logistic Regression	0.8636363636363636
5	Support Vector Machine (SVM)	0.967741935483871
6	Naive Bayes Classifier	0.05956813104988831
7	Decision Tree	0.8636363636363636
8	Neural Network Supervised (MLPClassifier)	0.797979797979798
9	Ridge Classification	0.8269230769230769

10	AdaBoost Classification	0.9146341463414634

Table 6. 2 precision score of our models

6.4 Accuracy score

One of the most basic assessment criteria that is commonly used to test classification models is the accuracy score. The precision score is determined by simply dividing the model's number of accurate predictions by the model's overall amount of predictions (can be multiplied by 100 to transform the result into a percentage). Generally, it can be expressed as:

$$Accuracy\ score = \frac{No.of\ correct\ predictions}{Total\ no.of\ predictions}$$

The six different classification models (Random Forest, KNN, Logistic Regression, SVM, Naive Bayes Classifier, Decision Tree) we created to check the accuracy score. We have used accuracy metrics by keras for ANN model (fig 5.2) and for six classifier model(fig 6.4) 'accuracy_score' method given by the scikit-learn package to do it in python.

Input:

```
from sklearn.metrics import accuracy_score
rfc_acc = accuracy_score(yTest, rfc_pred)
print("The accuracy score of Random Forest Classifier {}".format(rfc_acc))
tree_acc = accuracy_score(yTest, tree_Pred )
print("The accuracy score of Decision Tree {}".format(tree_acc))
knn_acc = accuracy_score(yTest, knn_pred )
print("The accuracy score of knn {}".format(knn_acc))
lr_acc = accuracy_score(yTest, lr_pred )
print("The accuracy score of Logistic Regression {}".format(lr_acc))
svm_acc = accuracy_score(yTest, svm_pred)
print("The accuracy score of svm {}".format(svm_acc))
NBC_acc = accuracy_score(yTest, NBC_pred)
print("The accuracy score of  Naive Bayes classifier {}".format(NBC_acc))
```

Output:

```
The accuracy score of Random Forest Classifier 0.9995962220427653
The accuracy score of Decision Tree 0.999403110845827
The accuracy score of knn 0.9995435553526912
The accuracy score of Logistic Regression 0.9991222218320986
The accuracy score of svm 0.9992977774656788
The accuracy score of  Naive Bayes classifier 0.9775113233383659
```

Figure 6. 4 Accuracy score of six different classification models by the scikit-learn package.

After evaluating the chapter 6.1 and 6.4 the accuracy score, we have found has appointed in table 6.3

#	Model	Accuracy score
1	Artificial Neural Network (ANN)	0.017450230196118355
2	Random Forest	0.9995962220427653
3	K-Nearest Neighbors (KNN)	0.9995435553526912
4	Logistic Regression	0.9991222218320986
5	Support Vector Machine (SVM)	0.9992977774656788
6	Naive Bayes Classifier	0.9775113233383659

7	Decision Tree	0.999403110845827
8	Neural Network Supervised (MLPClassifier)	0.9993153330290369
9	Ridge Classification	0.9988764439450862
10	AdaBoost Classification	0.9994733330992591

Table 6. 3 Accuracy score of models

6.5 F1 Score

One of the most common assessment metrics used for evaluating classification models is the F1 score or F-score. It can be generally described as the harmonic mean of both the accuracy and recall of a model. It is determined by dividing the model's precision product and recalling the value obtained by adding the model's accuracy and recall and eventually multiplying the result by 2. It is possible to express it as:

$$f1\ score = 2\left(\frac{(recall \times precision)}{(precision + recall)}\right)$$

The F1 score has calculated in python using the scikit-learn package's 'f1_score' function for six classifier model(fig 6.5) and for ANN model we have calculated manually(fig 5.2).

Input:

```
from sklearn.metrics import f1_score
rfc_f1 = f1_score(yTest, rfc_pred)
print("The f1  score  of Random Forest Classifier {}".format(rfc_f1))
tree_f1 = f1_score(yTest, tree_Pred )
print("The f1 score of Decision Tree {}".format(tree_f1))
knn_f1 = f1_score(yTest, knn_pred )
print("The f1 score of knn {}".format(knn_f1))
lr_f1= f1_score(yTest, lr_pred )
print("The f1 score of Logistic Regression {}".format(lr_f1))
svm_f1 = f1_score(yTest, svm_pred)
print("The f1 score of svm {}".format(svm_f1))
NBC_f1 = f1_score(yTest, NBC_pred)
print("The f1 score of  Naive Bayes classifier {}".format(NBC_f1))
```

Output:

```
The f1  score  of Random Forest Classifier 0.8620689655172413
The f1 score of Decision Tree 0.8172043010752688
The f1 score of knn 0.8555555555555556
The f1 score of Logistic Regression 0.6951219512195121
The f1 score of svm 0.75
The f1 score of  Naive Bayes classifier 0.11103400416377515
```

Figure 6. 5 F1 score of six different classification models by the scikit-learn package.

After evaluating the chapter 6.1 and 6.5 the f1 score that we have found has appointed in table 6.4

#	Model	F1 score
1	Artificial Neural Network (ANN)	0.04267265275120735
2	Random Forest	0.8685714285714285
3	K-Nearest Neighbors (KNN)	0.8555555555555556
4	Logistic Regression	0.6951219512195121
5	Support Vector Machine (SVM)	0.75
6	Naive Bayes Classifier	0.11103400416377515

7	Decision Tree	0.8172043010752688
8	Neural Network Supervised (MLPClassifier)	0.8020304568527918
9	Ridge Classification	0.5733333333333333
10	AdaBoost Classification	0.8333333333333334

Table 6. 4 F1 score of models

6.6 Matthews Correlation Coefficient (MCC)

The Matthews correlation coefficient (MCC) is often used in machine learning as an indicator of the consistency for multiclass and binary classifications. It takes into consideration true / false positives and negatives and is commonly known as a neutral measurement which could be used even though the groups are of quite various sizes. The MCC is in essence a correlation coefficient value between -1 and +1. A coefficient of +1 reflects a perfect prediction, 0 an average random prediction and -1 an inverse prediction. The figure is also known as the phi coefficient.

The MCC can be determined directly from the confusion matrix using the formula:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{\big((TP + FP) \times (FN + TN) \times (FP + TN) \times (TP + FN)\big)}}$$

Where,True Positive(TP), True Negative(TN), False Negative(FN) and False Positive(FP)
The MCC score has calculated in python using the scikit-learn package's "matthews_corrcoef" function for six classifier model (fig 6.6) and for ANN model we have calculated manually(fig 5.2).

Input:

```
from sklearn.metrics import matthews_corrcoef
rfc_MCC = matthews_corrcoef(yTest, rfc_pred)
print("The MCC  score  of Random Forest Classifier {}".format(rfc_MCC))
tree_MCC  = matthews_corrcoef(yTest, tree_Pred )
print("The MCC  score of Decision Tree {}".format(tree_MCC))
knn_MCC = matthews_corrcoef(yTest, knn_pred )
print("The MCC  score of knn {}".format(knn_MCC))
lr_MCC = matthews_corrcoef(yTest, lr_pred )
print("The MCC  score of Logistic Regression {}".format(lr_MCC))
svm_MCC  = matthews_corrcoef(yTest, svm_pred)
print("The MCC  score of svm {}".format(svm_MCC))
NBC_MCC  = matthews_corrcoef(yTest, NBC_pred)
print("The MCC  score of  Naive Bayes classifier {}".format(NBC_MCC))
nns_MCC = matthews_corrcoef(yTest, nns_pred )
print("The MCC score is Neural Network Supervised (MLPClassifier){}".format(nns_MCC))
RC_MCC = matthews_corrcoef(yTest, RC_pred )
print("The MCC score is Ridge Classification {}".format(RC_MCC))
AdBC_MCC = matthews_corrcoef(yTest, AdBC_pred )
print("The MCC score is AdaBoost  {}".format(AdBC_MCC))
```

Output:

```
The MCC  score  of Random Forest Classifier 0.8747121626683524
The MCC  score of Decision Tree 0.81809330663897
The MCC  score of knn 0.8587387603689554
The MCC  score of Logistic Regression 0.708352596990073
The MCC  score of svm 0.769449263019846
The MCC  score of  Naive Bayes classifier 0.21690316877855267
The MCC score is Neural Network Supervised (MLPClassifier)0.8016979038656407
The MCC score is Ridge Classification 0.6018960204490671
The MCC score is AdaBoost  0.836392916286224
```

Figure 6. 6 Matthews correlation coefficient (MCC) score of six different classification models by the scikit-learn package.

After evaluating the chapter 6.1 and 6.6 the Matthews correlation coefficient (MCC) score that we have found has appointed in table 6.5

#	Model	MCC score
1	Artificial Neural Network (ANN)	0.04267265647649765
2	Random Forest	0.8747121626683524
3	K-Nearest Neighbors (KNN)	0.8587387603689554
4	Logistic Regression	0.708352596990073
5	Support Vector Machine (SVM)	0.769449263019846
6	Naive Bayes Classifier	0.21690316877855267
7	Decision Tree	0.81809330663897
8	Neural Network Supervised (MLPClassifier)	0.8016979038656407
9	Ridge Classification	0.6018960204490671
10	AdaBoost Classification	0.836392916286224

Table 6. 5 Matthews correlation coefficient (MCC) score of models.

6.7 Confusion Matrix

Usually, a confusion matrix is a visualization of a classification model that indicates how well the model has predicted the outcomes as opposed to the original ones. Typically, the expected outcomes are stored in a variable that is then transformed into a correlation table. The confusion matrix is plotted by using the correlation table in the form of a heatmap.

Input:

```
from sklearn.metrics import confusion_matrix
ANN_cm=confusion_matrix(yTest,yPred_bool)
def CMatrix(ANN_cm, labels=['Normal','Fraud']):
    data = pd.DataFrame(data=ANN_cm, index=labels, columns=labels)
    data.index.name='TRUE'
    data.columns.name='PREDICTION'
    data.loc['Total'] = data.sum()
    data['Total'] = data.sum(axis=1)
    return data
CMatrix(ANN_cm)
```

Output:

PREDICTION TRUE	Normal	Fraud	Total
Normal	56864	0	56864
Fraud	98	0	98
Total	56962	0	56962

Figure 6. 7 Confusion Matrix of ANN model

Input:

```
# printing the confusion matrix
import seaborn as sns
LABELS = ['Normal', 'Fraud']
#conf_matrix = confusion_matrix(yTest, yPred)
plt.figure(figsize =(10, 10))
sns.heatmap(ANN_cm, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix of RANDOM FOREST")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Output:

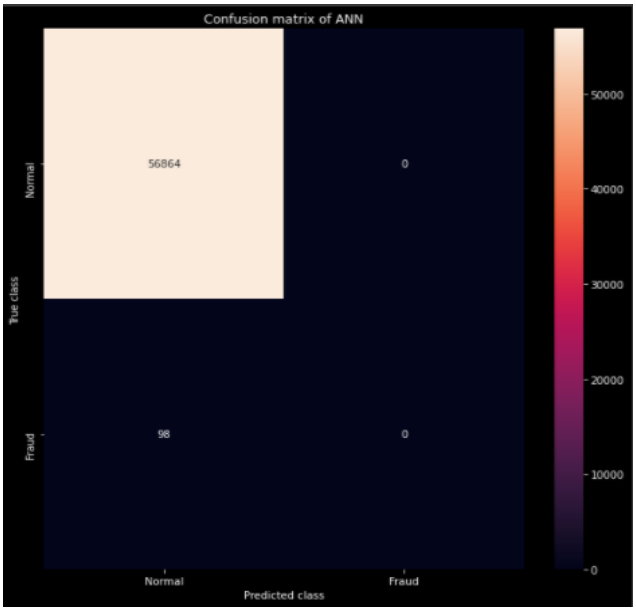


Figure 6. 8 Plotting Confusion Matrix heatmap of ANN model

Input:

```
rfc_cm = confusion_matrix(yTest, rfc_pred)
CMatrix(rfc_cm)
```

Output:

PREDICTION TRUE	Normal	Fraud	Total
Normal	56863	1	56864
Fraud	23	75	98
Total	56886	76	56962

Figure 6. 9 Confusion Matrix of Random Forest model

Input:

```
# printing the confusion matrix
import seaborn as sns
LABELS = ['Normal', 'Fraud']

plt.figure(figsize =(10, 10))
sns.heatmap(rfc_cm, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix of RANDOM FOREST")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Output:

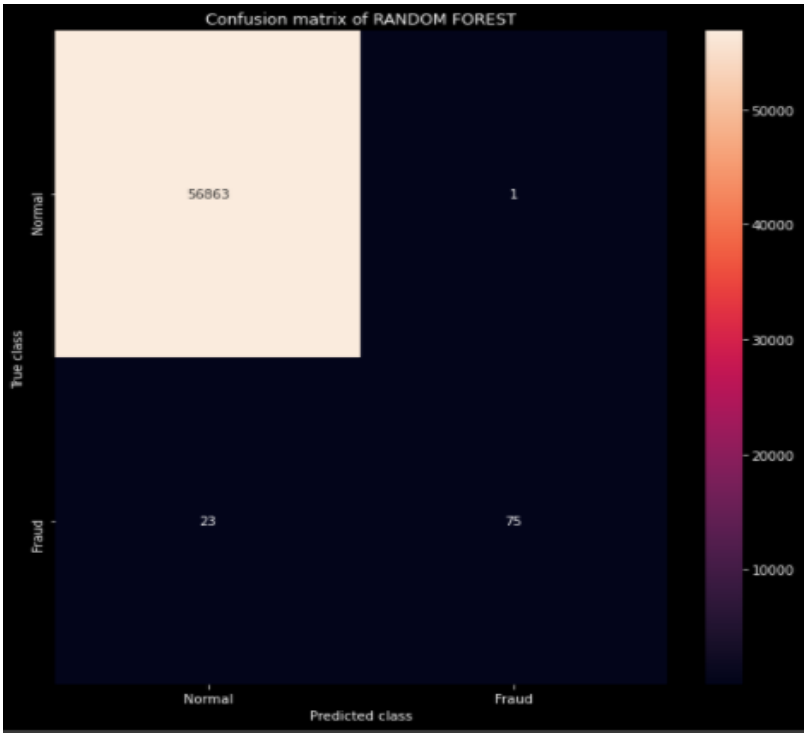


Figure 6. 10 Plotting Confusion Matrix heatmap of Random Forest model

Input:

```
knn_cm = confusion_matrix(yTest, knn_pred)
CMatrix(knn_cm)
```

Output:

PREDICTION TRUE	Normal	Fraud	Total
Normal	56859	5	56864
Fraud	21	77	98
Total	56880	82	56962

Input:

```
LABELS = ['Normal', 'Fraud']

plt.figure(figsize =(10, 10))
sns.heatmap(knn_cm, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix of KNN")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Figure 6. 11 Confusion Matrix of KNN model

Output:

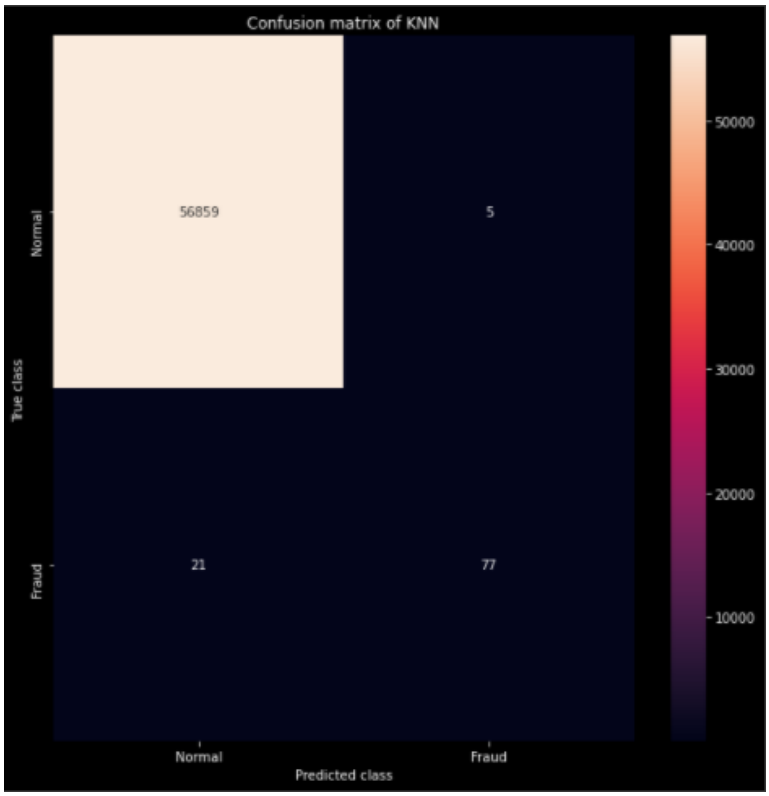


Figure 6. 12 Plotting Confusion Matrix heatmap of KNN model

Input:

```
lr_cm = confusion_matrix(yTest, lr_pred)
CMatrix(lr_cm)
```

Output:

PREDICTION TRUE	Normal	Fraud	Total
Normal	56855	9	56864
Fraud	41	57	98
Total	56896	66	56962

Figure 6. 13 Confusion Matrix of Logistic Regression model

Input:

```
LABELS = ['Normal', 'Fraud']

plt.figure(figsize =(10, 10))
sns.heatmap(lr_cm, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix of Logistic Regression")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Output:

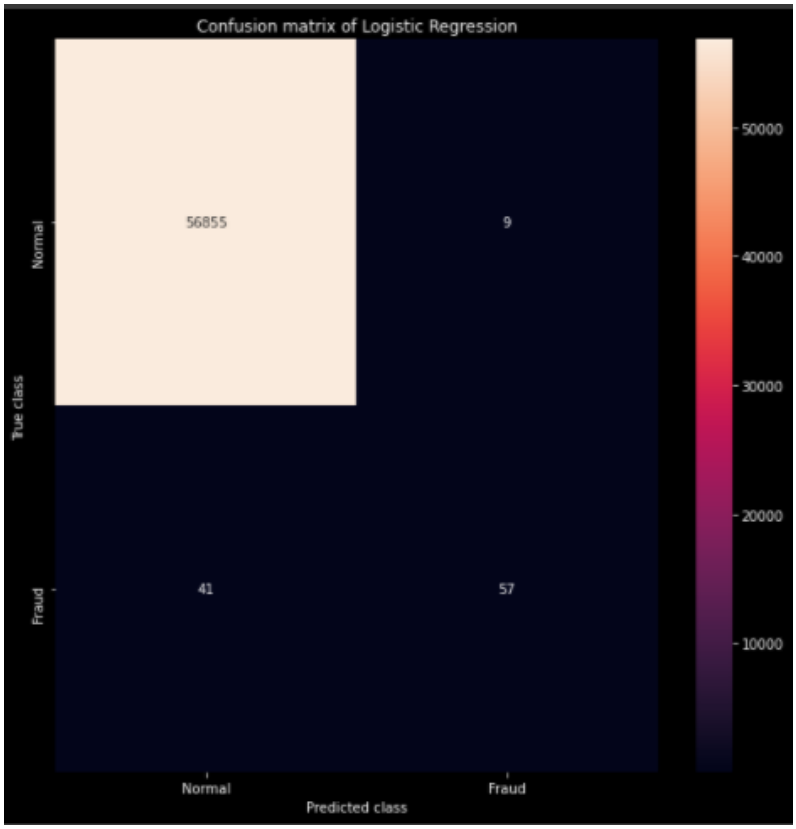


Figure 6. 14 Plotting Confusion Matrix heatmap of Logistic Regression model

Input:

```
svm_cm = confusion_matrix(yTest, svm_pred)
CMatrix(svm_cm)
```

Output:

PREDICTION TRUE	Normal	Fraud	Total
Normal	56862	2	56864
Fraud	38	60	98
Total	56900	62	56962

Figure 6. 15 Confusion Matrix of Support Vector Machine (SVM) model

Input:

```
LABELS = ['Normal', 'Fraud']
plt.figure(figsize =(10, 10))
sns.heatmap(svm_cm, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix of SVM")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Output:

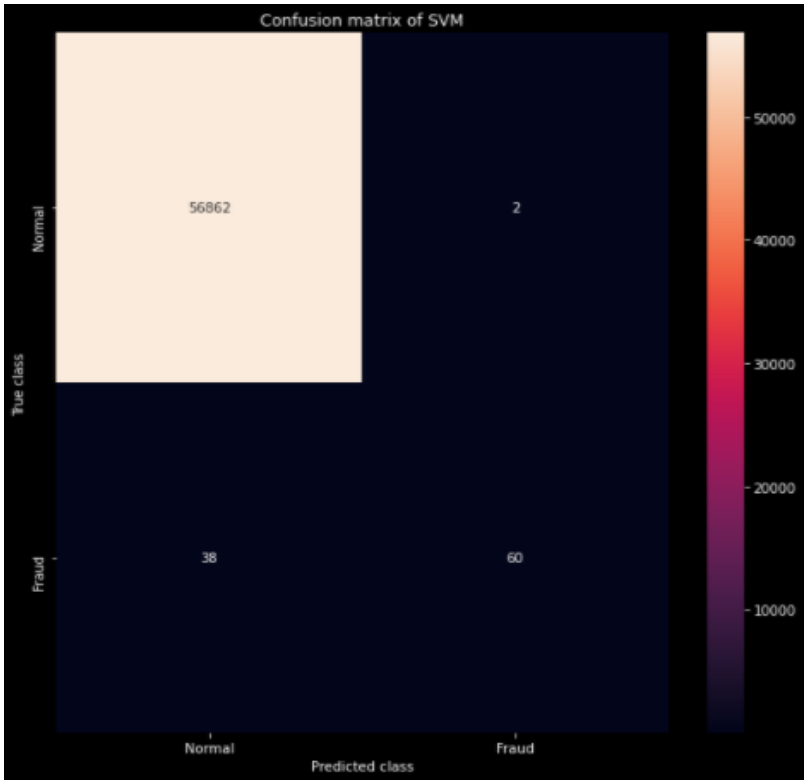


Figure 6. 16 Plotting Confusion Matrix heatmap of Support Vector Machine (SVM) model

Input:

```
NBC_cm = confusion_matrix(yTest, NBC_pred)
CMatrix(NBC_cm)
```

Output:

PREDICTION TRUE	Normal	Fraud	Total
Normal	55601	1263	56864
Fraud	18	80	98
Total	55619	1343	56962

Figure 6. 17 Confusion Matrix of Naive Bayes Classifier model

Input:

```
LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(yTest, NBC_pred)
plt.figure(figsize =(10, 10))
sns.heatmap(NBC_cm, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix of Naive Bayes")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Output:

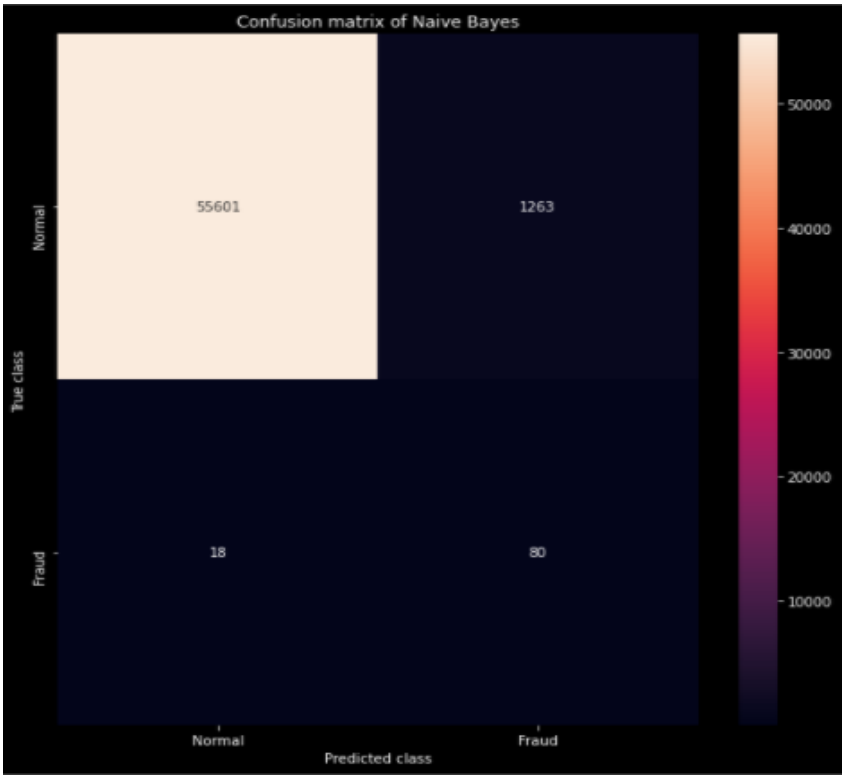


Figure 6. 18 Plotting Confusion Matrix heatmap of Naive Bayes Classifier model

Input:

```
from sklearn.metrics import confusion_matrix
nns_cm = confusion_matrix(yTest, nns_pred)

def CMatrix(nns_cm, labels=['Normal','Fraud']):
    data = pd.DataFrame(data=nns_cm, index=labels, columns=labels)
    data.index.name='TRUE'
    data.columns.name='PREDICTION'
    data.loc['Total'] = data.sum()
    data['Total'] = data.sum(axis=1)
    return data
CMatrix(nns_cm)
```

Output:

PREDICTION	Normal	Fraud	Total
TRUE			
Normal	56844	20	56864
Fraud	19	79	98
Total	56863	99	56962

Figure 6. 19 Confusion Matrix of Neural Network Supervised (MLPClassifier) model

Input:

```
from sklearn.metrics import confusion_matrix
nns_cm = confusion_matrix(yTest, nns_pred)

def CMatrix(nns_cm, labels=['Normal','Fraud']):
    data = pd.DataFrame(data=nns_cm, index=labels, columns=labels)
    data.index.name='TRUE'
    data.columns.name='PREDICTION'
    data.loc['Total'] = data.sum()
    data['Total'] = data.sum(axis=1)
    return data
CMatrix(nns_cm)
```

Output:

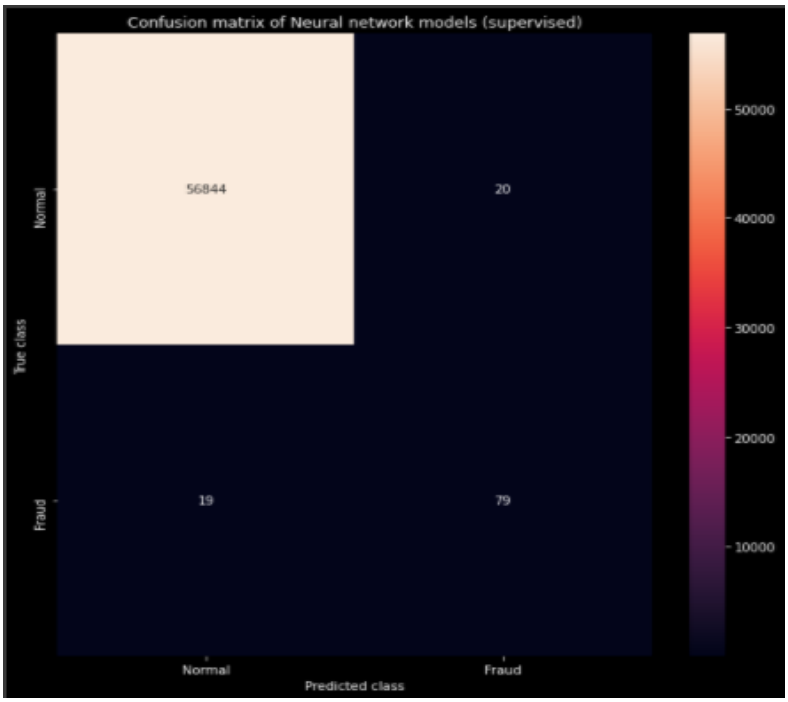


Figure 6. 20 Plotting Confusion Matrix heatmap of Neural Network Supervised (MLPClassifier) model

Input:

```
RC_cm = confusion_matrix(yTest, RC_pred)
CMatrix(RC_cm)
```

Output:

PREDICTION TRUE	Normal	Fraud	Total
Normal	56855	9	56864
Fraud	55	43	98
Total	56910	52	56962

Figure 6. 21 Confusion Matrix of Ridge Classification model

Input:

```
LABELS = ['Normal', 'Fraud']

plt.figure(figsize =(10, 10))
sns.heatmap(RC_cm, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix of RidgeClassifier")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Output:

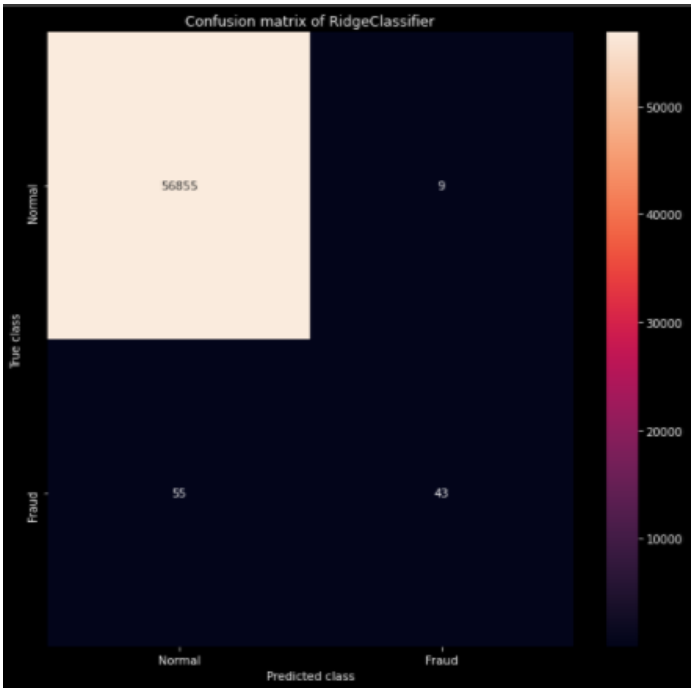


Figure 6. 22 Plotting Confusion Matrix heatmap of Ridge Classification model

Input:

```
AdBC_cm = confusion_matrix(yTest, AdBC_pred)
CMatrix(AdBC_cm)
```

Output:

PREDICTION TRUE	Normal	Fraud	Total
Normal	56857	7	56864
Fraud	23	75	98
Total	56880	82	56962

Figure 6. 23 Confusion Matrix of AdaBoost Classification model

Input:

```
LABELS = ['Normal', 'Fraud']

plt.figure(figsize =(10, 10))
sns.heatmap(AdBC_cm, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix of AdaBoost")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Output:

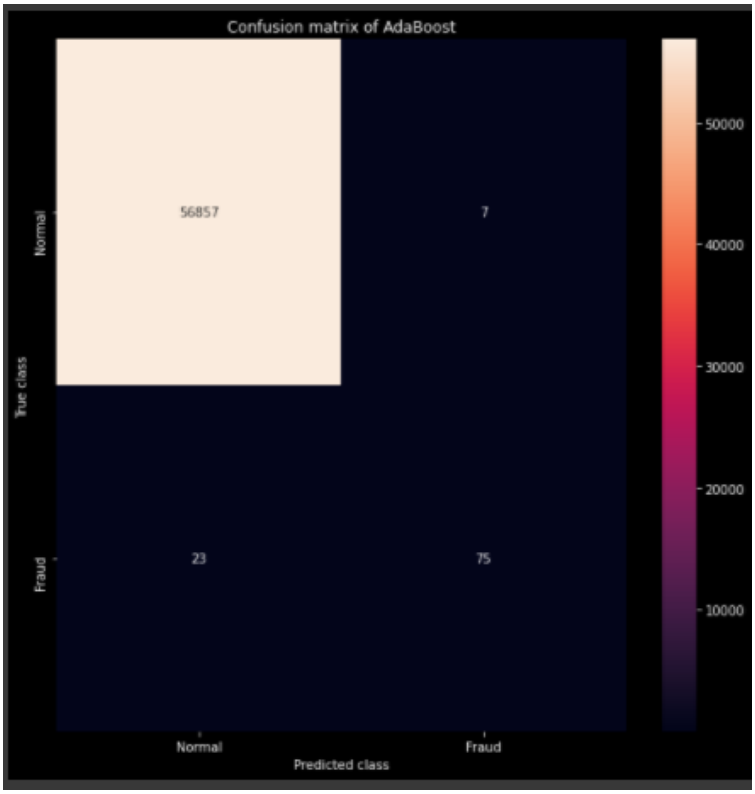


Figure 6. 24 Plotting Confusion Matrix heatmap of AdaBoost Classification model

Input:

```
tree_cm = confusion_matrix(yTest, tree_Pred)
CMatrix(tree_cm)
```

Output:

PREDICTION TRUE	Normal	Fraud	Total
Normal	56852	12	56864
Fraud	22	76	98
Total	56874	88	56962

Figure 6. 25 Confusion Matrix of Decision Tree model

Input:

```
LABELS = ['Normal', 'Fraud']

plt.figure(figsize =(10, 10))
sns.heatmap(tree_cm, xticklabels = LABELS,
            yticklabels = LABELS, annot = True, fmt ="d");
plt.title("Confusion matrix of Decision Tree")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Output:

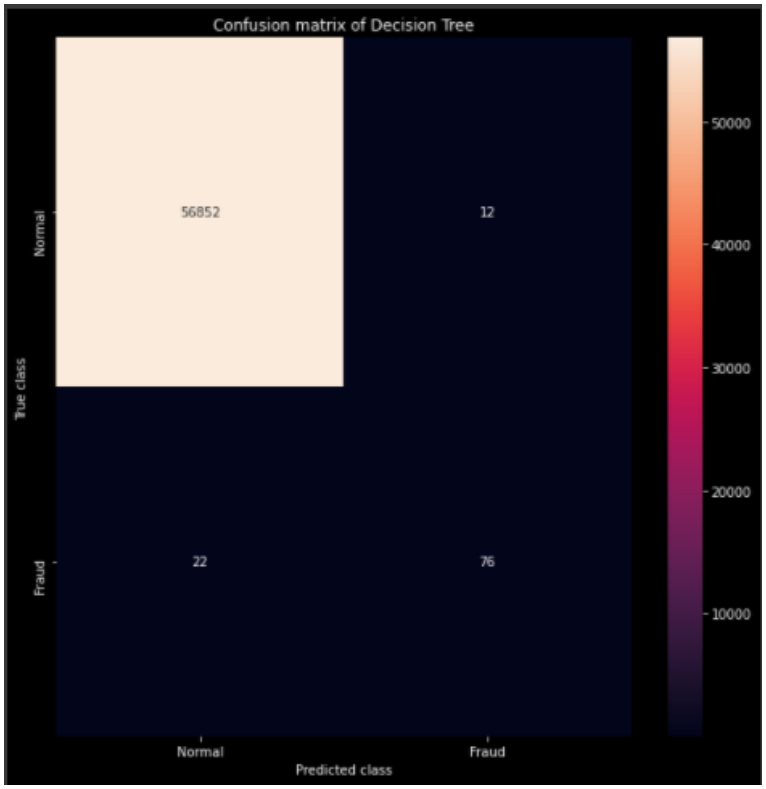


Figure 6. 26 Plotting Confusion Matrix heatmap of Decision Tree model

Understanding the confusion matrix:

Let's take the Decision Tree model's confusion matrix as an example. Check out the first row which is for transactions in the test set whose real fraud value is 0. The fraud value of 56864 of them is 0, as you can measure. And out of these 56864 non-fraud operations, 56852 of them were correctly predicted by the classifier as 0 and 12 of them as 1. This means that the real churn value was 0 in the test set for 56852 non-fraud transactions, and the classifier correctly predicted those as 0 too. We can assume that the non-fraud transactions were pretty well identified by our model.

The second row, let's look at it. It seems like there have been 98 transactions with a fraud value of 1. 76 of them were correctly predicted by the classifier as 1, and 22 of them wrongly as 0. It is possible to view the wrongly expected values as the model's mistake.

6.8 Lime

LIME, an algorithm that can faithfully clarify the predictions of any classifier or regressor by localizing it with an interpretable model.

```
import lime
from lime import lime_tabular

explainer = lime_tabular.LimeTabularExplainer(
    training_data=np.array(xTrain),
    feature_names=X.columns,
    class_names=['fraud', 'valid'],
    mode='classification'
)
```

Figure 6. 27 Generate LIME explanation

Input:

```
#ANN
def prob(data):
    print(data.shape)
    y_pred=classifier.predict(data).reshape(-1, 1)
    y_pred =(y_pred>0.5)
    print(np.array(list(zip(1-y_pred.reshape(data.shape[0]),y_pred.reshape(data.shape[0])))))
    return np.hstack((1-y_pred,y_pred))

i = 19
exp = explainer.explain_instance(X.loc[i,X.columns].astype(int).values, prob, num_features=5)
```

Output:

```
(5000, 30)
[[1 0]
 [1 0]
 [1 0]
 ...
 [1 0]
 [1 0]
 [1 0]]
```

Figure 6. 28 ANN model's array value for predict_proba

Input:

```
exp.show_in_notebook(show_all=False)
```

Output:

Figure 6. 29 ANN model interpretation by lime

Input:

```
#K-Nearest Neighbors (KNN)
exp = explainer.explain_instance(
    data_row=xTest[2],
    predict_fn=knn.predict_proba
)

exp.show_in_notebook(show_table=True)
```

Output:

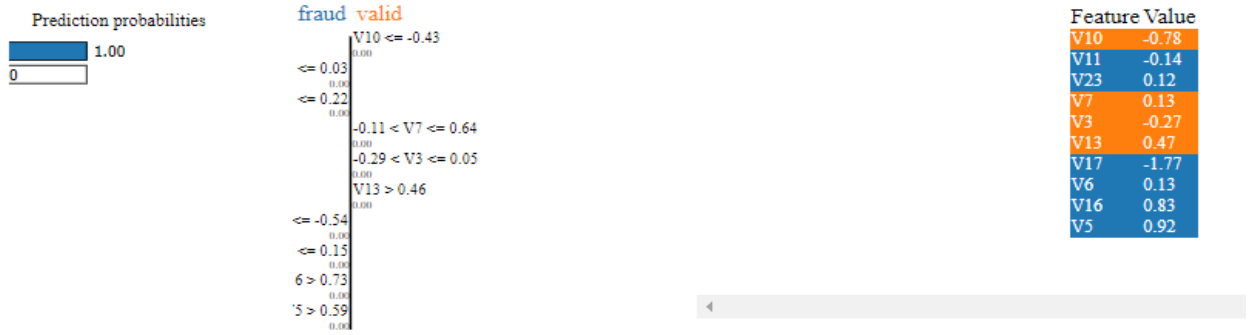


Figure 6. 30 KNN model interpretation by lime

Input:

```
#Random Forest
exp = explainer.explain_instance(
    data_row=xTest[2],
    predict_fn=rfc.predict_proba
)

exp.show_in_notebook(show_table=True)
```

Output:

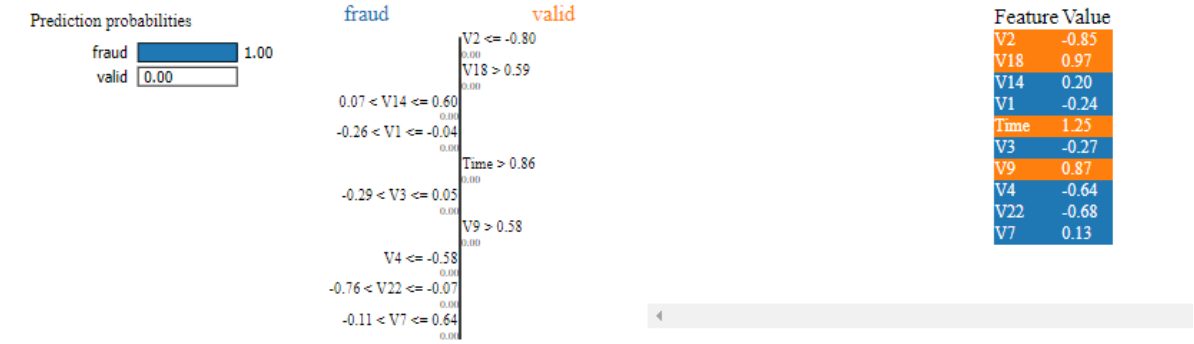


Figure 6. 31 Random Forest model interpretation by lime

Input:

```
#Logistic Regression
exp = explainer.explain_instance(
    data_row=xTest[2],
    predict_fn=lr.predict_proba
)

exp.show_in_notebook(show_table=True)
```

Output:

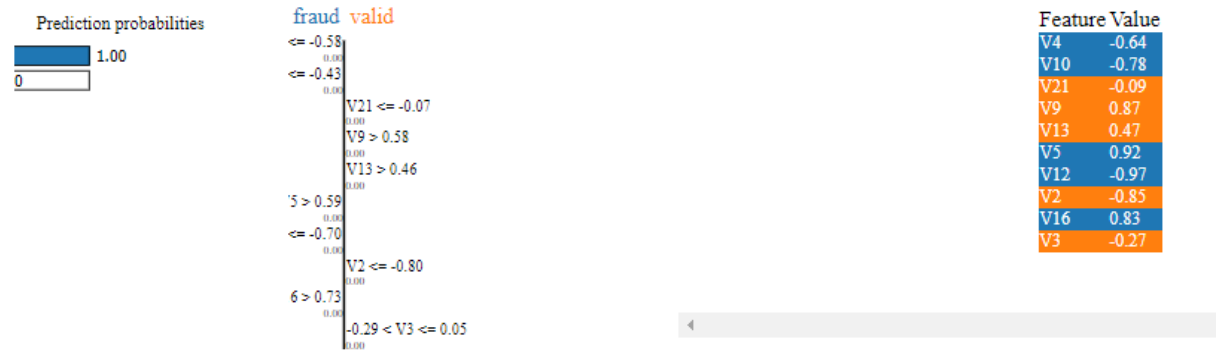


Figure 6. 32 Logistic Regression model interpretation by lime

Input:

```
#Support Vector Machine (SVM)
exp = explainer.explain_instance(
    data_row=xTest[2],
    predict_fn=svm.predict_proba
)

exp.show_in_notebook(show_table=True)
```

Output:

Figure 6. 33 SVM model interpretation by lime

Input:

```
#Naive Bayes Classifier
exp = explainer.explain_instance(
    data_row=xTest[2],
    predict_fn=NBC.predict_proba
)

exp.show_in_notebook(show_table=True)
```

Output:

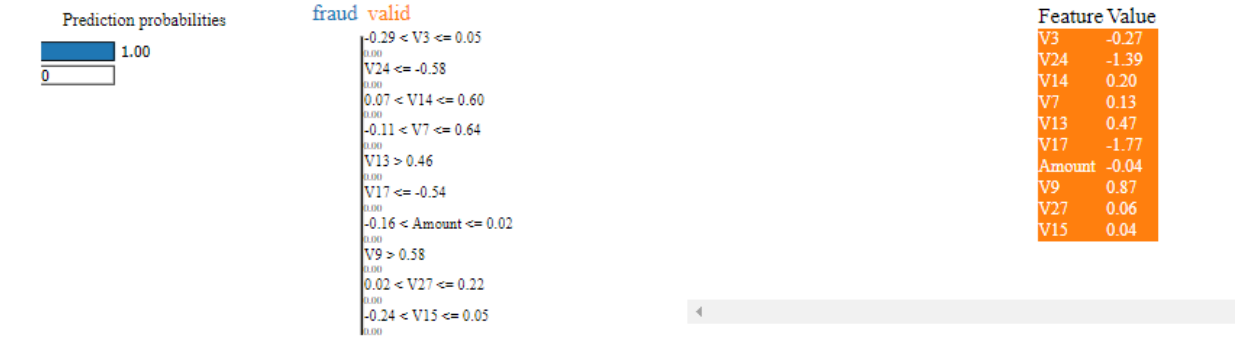


Figure 6. 34 Naive Bayes Classifier model interpretation by lime

Input:

```
#Decision Tree
exp = explainer.explain_instance(
    data_row=xTest[4],
    predict_fn=tree_model.predict_proba
)

exp.show_in_notebook(show_table=True)
```

Output:

Figure 6. 35 Decision Tree model interpretation by lime

Input:

```
#Neural Network Supervised (MLPClassifier)
exp = explainer.explain_instance(
    data_row=xTest[2],
    predict_fn=nns.predict_proba
)

exp.show_in_notebook(show_table=True)
```

Output:

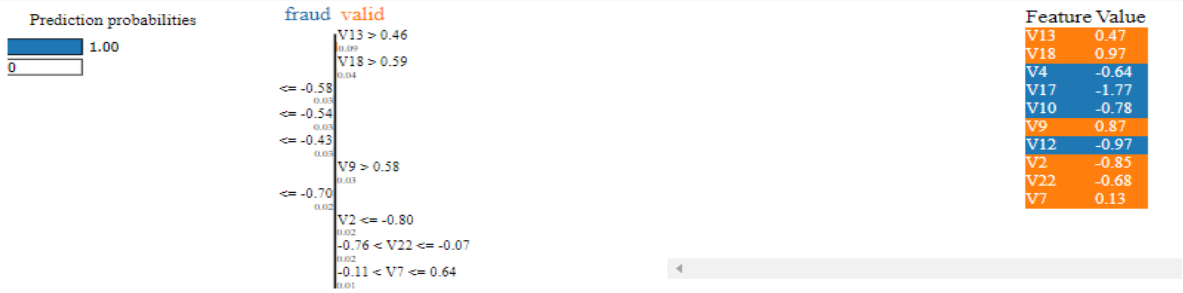


Figure 6. 36 Neural Network Supervised (MLPClassifier) model interpretation by lime

Input:

```
#Ridge Classification
def probs(data):
    print(data.shape)
    yy_pred=RC.predict(data).reshape(-1, 1)
    yy_pred =(yy_pred>0.5)
    print(np.array(list(zip(1-yy_pred.reshape(data.shape[0]),yy_pred.reshape(data.shape[0])))))
    return np.hstack((1-yy_pred,yy_pred))

i = 2
exp = explainer.explain_instance(X.loc[i,X.columns].astype(int).values, probs, num_features=5)
```

Output:

```
(5000, 30)
[[1 0]
 [1 0]
 [1 0]
 ...
 [1 0]
 [1 0]
 [1 0]]
```

Figure 6. 37 Ridge Classification model's array value for predict_proba

Input:

```
exp.show_in_notebook(show_all=False)
```

Output:

Figure 6. 38 Ridge Classification model interpretation by lime

Input:

```
#AdaBoost Classification
exp = explainer.explain_instance(
    data_row=xTest[2],
    predict_fn=AdBC.predict_proba
)

exp.show_in_notebook(show_table=True)
```

Output:

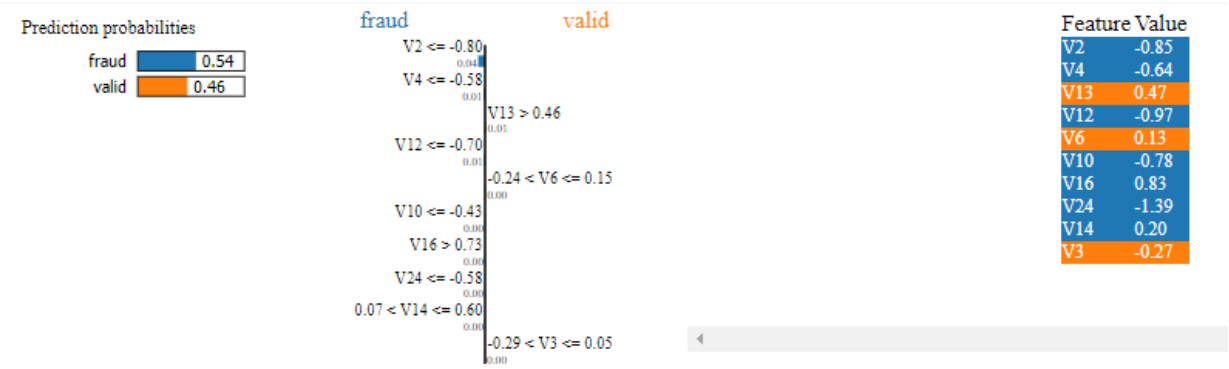


Figure 6. 39 AdaBoost Classification model interpretation by lime

Let's take the AdaBoost Classification model's lime as an example. The explanation consists of three parts:

1. The prediction probabilities appear to the left most section.
2. 10 main features are returned in the middle section. It would be in 2 orange/blue colors for the binary classification task. Attributes are valid for fraud and blue support classes in orange support class. The relative importance of these features is the float-point numbers on the horizontal bars.
3. Color coding across sections is consistent. It includes the actual values of the top five variables.

Chapter 7
Results and Discussion

In the training and testing sets, we have skewed our results. An under-sampling technique was used to balance the data. To compare our ten classification models, we used Accuracy, F1-Score, Recall, Precision, and Matthew's correlation coefficient (MCC).All classifier results and comparisons are shown in Table 7.1.

#	Model	Recall score	Precision score	Accuracy score	F1 score	MCC score
1	Artificial Neural Network (ANN)	0.7755101919174194	0.8539325594902039	0.017450230196118355	0.04267265275120735	0.04267265647649765
2	Random Forest	0.7755102040816326	0.9870129870129 87	0.9995962220427653	0.8685714285714285	0.8747121626683524
3	K-Nearest Neighbors (KNN)	0.7857142857142857	0.9390243902439024	0.9995435553526912	0.8555555555555556	0.8587387603689554
4	Logistic Regression	0.5816326530612245	0.8636363636363636	0.9991222218320986	0.6951219512195121	0.7083525969900 73
5	Support Vector Machine (SVM)	0.6122448979591837	0.9677419354838 71	0.9992977774656788	0.75	0.7694492630198 46
6	Naive Bayes Classifier	0.8163265306122449	0.0595681310498 8831	0.9775113233383659	0.1110340041 6377515	0.2169031687785 5267
7	Decision Tree	0.7755102040816326	0.8636363636363636	0.9994031108458 27	0.8172043010752688	0.8180933066389 7
8	Neural Network (supervised)	0.8061224489795918	0.7979797979797 98	0.9993153330290369	0.8020304568527918	0.8016979038656407
9	Ridge Classification	0.4387755102040816	0.8269230769230769	0.9988764439450862	0.5733333333333333	0.6018960204490671
10	AdaBoost Classification	0.7653061224489796	0.9146341463414634	0.9994733330992591	0.8333333333333334	0.8363929162862 24

Table 7. 1 Results and Comparisons of ten classification models

In table 7.1, we have got heights recall score 0.8061224489795918 from Neural Network (Supervised) model. And have got almost same recall score from ANN (0.7755101919174194), Random forest (0.7755102040816326) and Decision tree (0.7755102040816326) model. In addition have got the lowest value from Ridge Classification 0.4387755102040816.

From Random Forest model we have got heights precision score 0.987012987012987. Followed by SVM model 0.967741935483871 and lowest 0.797979797979798 from Neural Network (Supervised) model.

The heights and almost similar accuracy score we have got from models, Random Forest (0.9995962220427653) and KNN (0.9995435553526912). Afterwards, Decision Tree 0.999403110845827 and AdaBoost Classification 0.9994733330992591. Unexpectedly, From ANN model we have got the lowest accuracy score of 0.017450230196118355.

For F1 score, Random forest model gives the heights value of 0.8685714285714285. And then 0.8555555555555556 by KNN model. The lowest value from ANN model which is 0.8555555555555556.

Matthews Correlation Coefficient (MCC) score of 0.8747121626683524 is the heights value by Random forest model. Followed by KNN model (0.8587387603689554) which is the second heights value among the models. And 0.04267265647649765 is the lowest score by ANN model.

Chapter 8
Conclusions

This study examines the comparative performance in the binary category of imbalanced credit card fraud data of the Artificial Neural Network (ANN) Decision Tree, K-Nearest Neighbors (KNN), Logistical Regression, Support Vector Machine (SVM), Random Forest, Neural Network Supervised (MLPClassification), Ridge, AdaBoost and Naive Bayes. The reason why these ten techniques are investigated is because they were less comparable in past writings.

However, a further study is underway to compare other techniques for single and ensemble using our approach. The paper summarizes the contribution as follows:

1. Ten classifiers based on different machine learning techniques (ANN, Decision Tree, KNN, Logistic Regression, SVM, Random Forest, Neural Network Supervised (MLPClassifier), Ridge Classification, AdaBoost Classification, and Naive Bayes) are trained on real-world credit card transaction data, and their performance on credit card fraud detection is evaluated and compared using several relevant metrics.
2. A hybrid approach is used to sample the extremely imbalanced dataset, with the positive class being oversampled and the negative class being under sampled, resulting in two sets of data distributions.
3. The accuracy, f1, recall, precision, Matthew's correlation coefficient (MCC), and finally the confusion matrix are used to evaluate the ten classifiers' output on the two sets of data distributions. In addition, we used lime to interpret the models.

Classifier performance varies depending on the evaluation metric. The results of the experiment show that the Random forest outperforms the other models in all metrics except recall in the 10:90 data distribution. The effect of hybrid sampling on the performance of binary classification of imbalanced data is demonstrated in this analysis. Future research may look at meta-classifiers and meta learning approaches for dealing with highly skewed credit card fraud data. Other sampling methods' results may also be studied.

References

[1] Dataset: Credit Card Fraud Detection
Anonymized credit card transactions labeled as fraudulent or genuine(Version 3)
Available: https://www.kaggle.com/mlg-ulb/creditcardfraud

[2] The Sequential class
Available: https://keras.io/api/models/sequential/#pop-method

[3] Importance of Feature Scaling
Available:
https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html

[4] Scikit-learn Supervised learning
Available: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

[5] Analysis on credit card fraud detection methods
S. Benson Edwin Raj; A. Annie Portia
Available: https://ieeexplore.ieee.org/document/5762457/authors#authors

[6] Supervised Machine Learning Algorithms for Credit Card Fraudulent Transaction Detection: A Comparative Study
Sahil Dhankhad; Emad Mohammed; Behrouz Far
Available: https://ieeexplore.ieee.org/document/8424696

[7] "Why Should I Trust You?": Explaining the Predictions of Any Classifier
Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin
Available: https://arxiv.org/abs/1602.04938 [Submitted on 16 Feb 2016 (v1), last revised 9 Aug 2016 (this version, v3)]