DIFFERENTIAL EVOLUTION TECHNIQUE ON WEIGHTED VOTING STACKING ENSEMBLE METHOD FOR CREDIT CARD FRAUD DETECTION

by

KGAUGELO MOSES DOLO

submitted in accordance with the requirements for
the degree of

MASTER OF SCIENCE

In the subject of

COMPUTING

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: Prof Ernest Mnkandla

December 2019

SUMMARY

Differential Evolution is an optimization technique of stochastic search for a population-based vector, which is powerful and efficient over a continuous space for solving differentiable and non-linear optimization problems. Weighted voting stacking ensemble method is an important technique that combines various classifier models. However, selecting the appropriate weights of classifier models for the correct classification of transactions is a problem. This research study is therefore aimed at exploring whether the Differential Evolution optimization method is a good approach for defining the weighting function.

Manual and random selection of weights for voting credit card transactions has previously been carried out. However, a large number of fraudulent transactions were not detected by the classifier models. Which means that a technique to overcome the weaknesses of the classifier models is required. Thus, the problem of selecting the appropriate weights was viewed as the problem of weights optimization in this study.

The dataset was downloaded from the Kaggle competition data repository. Various machine learning algorithms were used to weight vote a class of transaction. The differential evolution optimization techniques was used as a weighting function. In addition, the Synthetic Minority Oversampling Technique (SMOTE) and Safe Level Synthetic Minority Oversampling Technique (SL-SMOTE) oversampling algorithms were modified to preserve the definition of SMOTE while improving the performance.

Result generated from this research study showed that the Differential Evolution Optimization method is a good weighting function, which can be adopted as a systematic weight function for weight voting stacking ensemble method of various classification methods.

DECLARATION

Name: Kgaugelo Moses Dolo

Student number: 58527141

Degree: Masters of Science (Computing)

Exact wording of the title of the dissertation as appearing on the copies submitted for examination:

Differential Evolution Technique On Weighted Voting Stacking Ensemble Method For Credit Card Fraud Detection.

I declare that the above dissertation is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

I further declare that I have not previously submitted this work, or part of it, for examination at UNISA for another qualification or at any other higher education institution.

_____ _____23/ 12/ 2019_____

SIGNATURE DATE

KEYWORDS

Differential Evolution; Weighted Voting; Stacking Ensemble Method; Class Distribution; Data Distribution; SMOTE; Safe-Level SMOTE; Machine Learning; Big Data; Credit Card Fraud.

Table of Contents

TABLE OF FIGURES

TABLE OF TABLES

ACRONYMS

ANN	**Artificial Neural Network**
CSV	**Comma-Separated Values**
DE	**Differential Evolution**
DT	**Decision Tree**
EDA	**Explanatory Data Analysis**
FN	**False Positive**
FP	**False Negative**
KNN	**k-Nearest Neighbour**
ML	**Machine Learning**
NB	**Naïve Bayesian**
NIBSS	**Nigerian Inter-Bank Settlement System**
NFL	**No Free Lunch**
P	**Probability**
PCA	**Principal Component Analysis**
SA	**South Africa**
SABRIC	**South African Banking Risk Information**
SL SMOTE	**Safe Level Synthetic Minority Oversampling Technique**
SMOTE	**Synthetic Minority Oversampling Technique**
SVM	**Support Vector Machine**
TN	**True Negative**
TP	**True Positive**
LII	**Legal Information Institute**

CITATION MANAGEMENT AND REFERENCE METHOD

References were managed electronically with the word document citation manager. Word document sources manager offers approximately 12 citation styles. For consistency, the University of Cape Town Harvard method (UCT Harvard) of referencing was used throughout this dissertation (De Jager & Steele, 2015)

1. Chapter 1: Introduction

The need for big data analytics in the banking sector is directly related to the need for complexity of analytical tasks for fraud detection. That is, as the need for big data analytics in the banking sector is becoming more and more important for supporting business operations and financial transactions (Altayar & Almoqren 2016, pp. 1 - 8), the need for complex analytical tasks for fraud detection is also becoming crucial for security of business operations and financial transactions. Banks normally derive data for data analysis from sources such as, but not limited to, credit card activities, loan repayments, loan applications and account transactions (Altayar & Almoqren 2016, pp. 1 - 8). For this reason, this study is focused on the credit card activities in order to determine fraudulent activities that negatively affect the banking sector.

The data derived from bank source systems are complex and cannot be processed by traditional data application systems (Altayar & Almoqren, 2016, pp. 1 - 8). Big data is a term that refers to a complex dataset that cannot be stored and processed using traditional data processing application systems (Flood, et al. 2016, pp. 1 - 20). Banks in South Africa are making huge investments in the adoption of big data and the move from data analytics to data insights (TCI 2015). Data insights refers to the use of advanced analytics methods that deal with complex analytical problems and are capable of predicting future outcomes (i.e. predictive analytics). Predictive analytics refers to many types of statistics and data mining methods that analyse current and historical facts for the classification of unknown future events (Monk 2016, pp. 1 - 3).

The accurate and comprehensive methods used for filtering algorithms that perform data analysis for a given dataset of historical credit card transactions of the customers play a crucial role in determining the efficiency of algorithms that perform data mining and predictive data analysis (Mitik, et al. 2016, pp. 552 – 559; Wang, et al. 2010, pp. 215 - 218). Identifying fraudulent factors associated with credit card transactions in the banking sector is one of the important tasks to perform that enables the data mining activity to uncover fraudulent patterns (Babu & Rajeshwari 2016, pp. 439 - 444). The NO-FREE-LUNCH (NFL) theory suggests that no machine learning method can perform well for all measures in every application (Macready & Wolpert 1996, pp. 67 - 82). Therefore, an evaluation of algorithms is essential for the quality of machine

learning methods for revealing predictive data insights to prevent fraudulent transactions of credit cards in South Africa (SA).

Fraud data sourced from the South African Banking Risk Information (SABRIC) suggest that almost 4.5 billion was lost in South Africa (SA) to credit and debit card fraud since the year 2010. In analysing the 2010-2016 period, a 3.5% (44.5% in 2015 and 48.0% in 2016) was seen in credit card fraud loss in SA (Writer, 2015). Banking fraud in the continent is not confined to SA. For example, data released by the Nigerian Inter-Bank Settlement System (NIBSS) indicates that malicious transactions in the banking sector peaked in 2016 (John, et al. 2016, pp. 1186 - 1191). Since SA has the most advanced banking sector in the continent, it goes without saying that more complex analytical tasks for fraudulent transactions detection are required to expose fraudulent activities within the SA banking sector.

1.1. Background

According to the Legal Information Institute (LII), fraudulent credit card transactions are a form of identity theft whereby unauthorized users take credit card information of victims with the sole aim of removing funds from the accounts of the victor or to charge purchases to the accounts of the victims. The credit card fraud level has escalated from being performed by just the fraudsters to being performed by fraud rings that use complex and advanced strategies that take over the control of accounts and to perform frauds that are not noticeable (John, et al. 2016, pp. 1186 - 1191). In the world of effortless expenditure, this leads to an accelerated increase in online transaction fraud.

More fraudulent transactions performed in a day in relation to more non-fraudulent transactions result in less skewed data distribution, which implies that the chances of frauds being noticeable are high. Thus, since fraudsters use advanced methods to commit crime in a manner that is not noticeable (John, et al. 2016, pp. 1186 - 1191), it implies that less fraudulent activities are performed than non-fraudulent activities, and in the context of big data this would result in a highly skewed distribution. Thus, more complex and efficient analytical tasks to handle highly skewed distribution are required.

Credit card fraud has caused serious damage to both service providers and the credit card users, and was predicted to get worse in future (Pushpa & Malini 2017, pp. 255 -

258). Thus, to handle the highly skewed big data for classification or prediction of future credit card fraudulent transactions, more advanced machine learning techniques to reduce variance and biasness in data are required in order to adequately predict future values of credit card transactions. Given the large number of datasets available, development of methods for the identification of fraudulent groups and exploration of behavioural patterns of fraudulent groups have become very important for purpose of data mining (Yu, et al. 2017, pp. 1 - 6).

Descriptive and predictive data analysis provides a complete solution for understanding the nature and scope of the data of financial institutions sourced from different systems. Descriptive data analysis is more focused on the question of “what has happened?” (Patwardhan 2016). Knowing what has happened without knowing what could happen, sets knowledge limitations and biasness of financial institutions to make informed decisions. On those group, the need for predictive data analysis is justified since it answers the question of “what could happen?” (Patwardhan 2016).

When building a classification data model to solve the problems of descriptive and predictive data analysis with a dataset that is highly skewed, the best-fit line that separates the classes of data would try to balance the dataset by drawing a line that represents the average of the dataset of the class of the minority data observations and the class of the majority data observations. However, given a lower number of fraudulent transactions relative to non-fraudulent transactions, the average dataset would be in favour of the non-fraudulent transaction class. Therefore, to control the best-fit line to the right position that equally represent both classes, an oversampling of minority class to generate new synthetic instances is required.

Synthetic Minority Oversampling Technique (SMOTE) is a sampling method that oversamples the minority class by computing median features vectors between nominal features sample and its potential nearest neighbours through Euclidean distance of standard deviations (Chawla, et al. 2002, pp. 321 - 357). Duplicating minority training samples to reduce biasness of data distribution introduces high variance of data distribution. Therefore, SMOTE is an important over-sampling method that reduces variance of data distribution. The synthetic instances generated influence a decision-making model to create small and less specific decision regions.

With that said, positive influence can be learned far better in general regions than positive instance subsumed around negative instance. This means that the SMOTE method suffers from a problem of generalization whereby the region of a majority class is blindly generalized without considering the minority class (Bunkhumpornpat, et al. 2009, pp. 475 - 482; Meidianingsih, et al. 2017, pp. 1167 - 1171). The generalization challenge associated with the SMOTE method is normally visible in places of highly skewed class distribution since the class of the minority data observations is thinly scattered in relation to the class of the majority data observations. As a result, the probability of class mixture is very high. To keep the SMOTE method efficient and effective, an improvement of the algorithm is required.

Borderline-SMOTE is an oversampling technique designed to improve the performance of SMOTE oversampling method (Bunkhumpornpat, et al. 2009, pp. 475 - 482; Gosain & Sardana 2017, pp. 79 - 85). The performance is improved by separating the positive instances into three regions namely borderline ($\frac{1}{2}K \leq num < K$), noise ($num = K$), and safe ($0 \leq num < \frac{1}{2}K$) (Bunkhumpornpat, et al. 2009, pp. 475 - 482; Gosain & Sardana 2017, pp. 79 - 85). The three regions are separated while considering the instances that are negative in the k-Nearest Neighbours. The borderline SMOTE uses the same oversampling method as SMOTE.

However, borderline SMOTE oversamples only the borderline data observations of the minority class rather than oversampling the entire data observations of the minority class. Logically, the two consecutive data observations are obviously not different are instead separated into two regions (borderline and noise) whereby the first data observation is selected for oversampling and the other data observation is declined oversampling.

That being so, the Safe-Level-SMOTE method is the SMOTE method that creates safe-level synthesis of minority class (Bunkhumpornpat, et al. 2009, p. 475 - 482; Meidianingsih, et al. 2017, pp. 1167 - 1171; Gosain & Sardana 2017, pp. 79 - 85). The synthetic instances are placed closer to the safe level; this means that the safe level closer to K is nearly noise. The safe level is the number of positive data observations within k-Nearest Neighbour but not equal to k-Nearest Neighbour (Bunkhumpornpat, et al. 2009, pp. 475 - 482).

The Safe-Level SMOTE is a promising algorithm for achieving a positive impact. For this reason, this algorithm will be used in this study to oversample the data observations of the minority class of the dataset and thus control the best fit line to an optimum place that equally represent malicious transactions and non-malicious transactions. A dataset with equal representation of malicious transactions and non-malicious transactions makes it easier for a classification model to learn data of the two classes well.

Stacking is an ensemble method that uses different machine learning algorithms to generate new dataset that is used in a combiner machine learning method (Smolyakov 2017). The combiner machine learning voting method is computed to predict the output. Majority voting is when every model makes a prediction of a transaction and the class value with more votes is produced as final output. Weighted voting is voting that is undertaken by counting the predictions of better models to predict the output of each transaction. In this study, it is suggested that weighting voting of the classification models requires an optimization of weights to produce good results.

Differential Evolution algorithm is a subset of evolution programming designed for continuous domain optimization problems (Brest, et al. 2006, pp. 646 - 657; Srinivas, et al. 2018, pp. 216 - 217; Madathil, et al. 2017, pp. 1 - 5). Since it is normal for the base models of the stacking ensemble method to have weak classifiers, Differential Evolution is used in this study to learn the weak classifiers in each evolution stage and correct the weights (Madathil, et al. 2017, pp. 1 - 5). Differential Evolution has an advantage over genetic algorithms because it is fast, robust, easy to use, and simple in structure (Brest, et al. 2006, pp. 646 - 657; Srinivas, et al. 2018, pp. 216 - 217; Madathil, et al. 2017, pp. 1 - 5).

Genetic algorithms use selection mechanisms to produce a sequence of population, and thereafter use mutation and crossover as search mechanism. Unlike Genetic algorithms, the only search mechanism of Differential Evolution is mutation, and the search is guided towards the prospective regions inside a feasible region through the use of selection (Kalajac, et al. 2018, pp. 883 - 886). The main difference between Differential Evolution and Genetic algorithms is that whereas Genetic algorithms are more dependent on a mechanism of probabilistic, a crossover, and an exchange of

information in solutions to better locate solutions, Differential Evolution strategies are dependent more on mutation as a mechanism for the primary search.

When features of customer's data on historical credit card transactions that are relevant for data mining are available, the explanatory data analysis plays a crucial role when determining the efficiency of the dataset in the data mining and the predictive data analysis (Todd & Harvey 2015, pp. 474 – 489; Liu & Dash 1997, pp. 131 – 156; Chezian & Devi. 2016, pp. 161 - 165). That being so, feature selection and dimension reduction are important for minimizing the complexity of the predictive data analysis models to improve the classification speed.

Fraud detection refers to the process of detecting fraudulent activities where no prior tendency of fraud exists (John, et al. 2016, pp. 1186 - 1191). To improve the performance of predictive data analysis methods, more advanced performance optimization methods for boosting performance while retaining low variance and low biasness of data are required for the detection of credit card fraudulent activities and thus exposing of hidden fraudulent transactions. The performance optimization methods that can adequately optimize parameters of predictive data analysis methods in massive data are becoming more important than ever due to the amount of data being produced in the current era.

The most challenging part of fraud detection involves uncovering fraudulent activities in a complex dataset that has a high number of non-fraudulent transactions and a low number of fraudulent transactions (Pushpa & Malini 2017, pp. 255 - 258). The dataset complexity in an environment of rapidly growing data introduces inefficiency in fraud detection systems, and the inefficiency acts as an agent that motivates and accelerates the rate of increase of fraudulent transactions in the banking sector. Hence the banking sector must make a huge investment in complex and efficient analytical systems that detect fraud activities to prevent losses.

1.2. Problem Statement

According to the Trade Conferences International (TCI 2015), SA banks are making huge investments in big data adoption and are intended on moving from data analytics to data insights by 2020. Notwithstanding available data analytics models for the detection of fraudulent transactions, the rate at which fraudsters continue to commit

fraud is growing continuously. This implies an existence of means that there is an unknown gap between knowledge of customers and techniques that fraudsters take advantage of when committing fraud.

Currently available systems of fraud detection are capable of uncovering fraudulent activities after one, two or more fraudulent transactions have been performed (Wang, et al. 2015, pp. 354 - 359). This suggests that the advanced analytics techniques require strategies that overcome the weaknesses of classifier models. Stacking is an ensemble method that uses a variety of base models to predict the output of each transaction (Smolyakov 2017).

The stacking ensemble method is an important technique that combines various classifier models such that a weak classifier model for a particular instance is assisted by strong classifier models through weighted voting of a class value (Zirpe & Joglekar 2017, pp. 1 – 5; Verma & Mehta 2017, pp. 155 – 158; Cheng, et al. 2012, pp. 755 - 759). The weights for classifier models per transaction must be high for a class that performs well, and low for a class that does not perform well. Reasoning from this fact, the selecting of the appropriate weights of votes for each class per classifier is very important. On this basis, the problem of selecting the appropriate weights for voting the correct output class of each transaction is viewed as the problem of weights optimization in this research study.

For detection systems, the standard machine learning algorithms that achieve high accuracy tend to classify a higher percentage for the class with the majority of data observations compared to a percentage of the class with the minority of data observations (Mitik, et al. 2016, pp. 552 – 559; Dehghan, et al. 2017, pp. 124 - 134). This means there are overlapping data points between the class with minority data observations and the class with majority data observations. Oversampling the minority class by duplicating the minority class data observations when there are overlapping data observations may not solve the problem of high skewedness of class distribution. Using SMOTE oversampling method to create the synthetic observations of the class with minority data observations when there are overlapping data points of the class with majority data observations result in synthetic instance generated from a fraudulent transaction and non-fraudulent transaction. Therefore, a need exists for the

development of an oversampling method that generates synthetic instances at the safe level of the minority class.

1.3. Research Questions

Since the South African banking sector is migrating from data analytics to data insights, this research study intends to explore the following main research question:

> *Is the differential evolution optimization method a good approach for defining the weighting function to predict the outcome of future credit card fraudulent transactions?*

The sub research questions that support the main research question of this research study are as follows.

1. Does the Safe-Level-SMOTE oversampling method (on the minority classes) when used in combination with an under-sampling method eliminates duplicate data samples of the majority class have a positive impact on reducing the high skewedness of the class distribution than the SMOTE oversampling method (on the minority classes) when used in combination with the under-sampling method (also eliminates duplicate data samples on the majority class)?

2. Does Safe-Level-SMOTE oversampling method and the under-sampling method (that eliminates duplicate data samples of the class with majority data observations) reduce or eliminate the problem of overlapping data samples between fraudulent and non-fraudulent classes?

3. Does manual stacking of feature selection methods that are diverse on the Principal Component Analysis (PCA) transformed features to hide the actual feature names, select enough good features to solve the problem than just selecting a feature engineering technique?

4. Does the stacking of supervised base models have an impact on predictive accuracy given the under-sampling technique to remove the duplicate data

observations on the majority class, and the Safe-Level-SMOTE oversampling method?

5. Will the proposed data model for fraud detection efficiently detect credit card fraudulent activities?

1.4. Research Aim

This study aims to develop and optimize a detection method for credit card fraudulent transactions that is efficient and effective in the detection of malicious behaviours within the minimum number of attempts or the first time the fraudster attempts to commit fraud. The developed model is also expected to overcome the weaknesses of the classifier models that fraudsters take advantage of when committing credit card fraud.

The continuation of credit card fraud results in monetary losses for customers, merchant stores, investors, the banking sector and various other parties. Therefore, it is envisaged that the development of such a model will add value to the knowledge economy of SA and the banking sector through successful implementation of data insights. Furthermore, the development of an efficient and effective method for the detection of credit card fraud will serve to discourage the continuation of activities association with malicious credit card crime. In particular, this research study will have an impact on the successful implementation of data insights.

1.5. Research Objectives

The main research objective is to obtain optimum data analytics algorithms that perform data insights to reveal fraudulent data patterns in order to prevent future fraudulent transactions. The sub objectives are:

1. To determine whether if Safe-Level SMOTE oversampling method when used in combination with under-sampling method has a positive impact on reducing the high skewedness of the class distribution than the SMOTE oversampling method when used in combination with the under-sampling method. This objective will be implemented by running a SMOTE algorithm

and a Safe-Level SMOTE algorithm on the class of minority data observations, and a down-sampling algorithm on the class of majority data observations.

2. To investigate whether the Safe-Level SMOTE oversampling technique and the under-sampling technique reduce or eliminate the problem of overlapping data observations of fraudulent and non-fraudulent classes. This objective will be implemented by running a Safe-Level SMOTE on the class of minority data observations and an under-sampling algorithm on the class of majority data observations.

3. To determine whether manual stacking of diverse feature selection methods on the PCA transformed features to solve the problem than feature selection method. This objective will be implemented by running univariate selection algorithm, recursive feature elimination algorithm, and feature importance algorithm on the entire dataset.

4. To investigate how the stacking of supervised machine learning algorithms have an impact on predictive accuracy, considering the down-sampling technique that eliminate duplicate data samples on the class of majority data observations, and the SMOTE oversampling method. This objective will be implemented by running both the differential evolution optimization and stacking algorithms on the classifier models.

5. To determine the efficiency of the proposed data model for fraud detection in the detection of activities relating to credit card fraud. This objective will be implemented by running both the differential evolution optimization and the stacking algorithm on the classifier models.

1.6. Limitation

The limitation of this research study is the inability of the researcher to use a largely sized data sample drawn from the population of credit card users in SA banks. The inability of the researcher to use a dataset implies that large data cannot be applied in

this research study. For confidentiality and security reasons in the SA banking sector, this research study is unable to get credit card dataset from any bank in the country for experimental purposes. However, the study will use the credit card dataset designed for experimental purposes, which is obtained from the Kaggle machine learning data repository (kaggle 2018).

Due to field names having been transformed for hiding the actual field name of the dataset, the data mining of new knowledge and for uncovering the unknowns becomes challenging. Although correlation among different fields can be performed and the findings can be interpreted, the findings cannot be understood for further data analysis due to hidden field names. On that premise, the explanatory data analysis section presented in this study, which is of interesting to this research study, is rudimentary and cannot therefore be analysed further to derive meaningful interpretation of the data. However, the dataset is stable enough to achieve the desired research objectives.

Due to different cultures found in SA banks and the approach adopted by the researcher for conducting the research, the applied techniques may produce different results when experimenting with credit card dataset obtained from Kaggle machine learning data repository. However, traditional data mining algorithms for uncovering new knowledge and predictive data analytics algorithms for predicting future outcomes are more likely to remain the same if they are used appropriately. Consequently, the data model proposed in this research study must be edited appropriately to produce the desired outcomes.

1.7. Ethics Considerations

Considering the pertinent banking sector rules, regulations, and norms that govern the industry, the data mining and prediction data model in the banking sector may raise ethical and legal questions. This research study is therefore focused on computational analysis of credit card dataset designed for experimental purpose from the Kaggle machine learning data repository, and would therefore not breach any research ethics for the banking industry and its customers.

Although this research study does not involve experimentation using humans and animal subjects, ethical approval for the undertaking of this research study was

nevertheless requested in writing from the Research Ethics Committee of the University of South Africa (UNISA). To get the gist of the approach adopted by the researcher for executing this research study, the reader is referred to sub-sections 1.4 and 5.1 relating to research aims and methodological approach and experimental design respectively. The dataset used for this experiment is credit card data that was anonymised by Principal Component Analysis (PCA) method for hiding the private and confidential information of the bank users.

1.8. Significance of the Research

The findings of in this study contribute to the knowledge economy of South Africa and the banking sector in general. Safe-Level SMOTE for the detection of malicious activities relating to credit card fraud is significant in a case of overlapping class data because oversampling of the minority class in a highly skewed distribution with overlapping class data does not achieve 50:50 binary class distribution. That is to say, one problem is being solved by introducing another problem.

A method for ensemble feature selection for unknown features that are transformed to hide the original meaning is significant to guide conclusion in terms of features selected, which are suggested by this study.

An ensemble data model with diverse base models that is based on weighted voting method is significant since a weak base model cannot be given the same weighting as a strong base model. The optimization of the weighted voting method of base model is also significant since an improvement of predictive accuracy needs to be undertaken. The optimization of the weighted voting method is performed by using the differential evolution algorithm.

The biasness reduction of data distribution within a class and a class distribution plays a significant role since duplicate data samples are eliminated while information about the dataset is preserved as much as possible.

1.9. Layout of the Dissertation

The remainder of the dissertation presents a review of the relevant literature in Chapter 2. This is followed by Chapter 3, which discusses the research methodology adopted for the execution of this research study. Therefore, Chapter 4 provides details of the

algorithm design for this research study. In Chapter 5 data is analysed ending with Chapter 6, which presents the conclusion of the study. Following an analysis of the relevant data, the dissertation ends with chapter 6, whereby the conclusion of the study is presented.

2. Chapter 2: Literature Review

In this section, literature relevant to the research study is reviewed. The section starts by giving background information on big data, and how big data is defined in the current era. Thereafter, reference is made to different types of credit card fraud experienced on a daily basis by the banking industry. In addition, the information and personal data security as an enforcement in Section 14 of the Constitution of South Africa (1996) is discussed. Finally, the state-of-the-art credit card fraud detection techniques for preventing fraudulent transactions are also chronicled before discussing the challenges associated with credit card fraud detection techniques.

2.1. Background on Big Data

Big data described as large amount of data depending on who is discussing it. The type of data regarded by the likes of Google or Amazon as being huge may be totally different from the huge data of a small or medium sized organization. The history of big data does as far as 1663 when John Graunt dealt with huge amounts of data when studying the bubonic plague (Keith 2017). To describe and understand big data in this study, four V's (namely Velocity, Volume, Variety and Veracity) are used.

2.1.1. Volume

The volume describes the size of dataset (Nyikes & Rajnai 2015, pp. 217 – 222; Trelewicz 2017, pp. 1 - 3). For big data, volume describes the large sized data collected for uncovering the unknown factors that will enable business questions to be answered. In the world of effortless expenditures, the high number of credit card users translates into a higher number of daily translations. As a result, the concomitant reporting and data analysis of the daily data credit card transactions tallied over a period of a month is also high.

Since banks in South Africa have been operating for years, the amount of credit card data collected over the years is just too complex to be handled by traditional data processing applications. Therefore, complex data analytical tools are required for the processing of the processing of this type of data.

2.1.2. Velocity

The velocity is mostly associated with the rate at which data is generated, accumulated or must be processed (Nyikes & Rajnai 2015, pp. 217 – 222; Trelewicz 2017, pp. 1 - 3). Given the high number of credit card users, it is realistic to think that a substantial majority of these users have mobile phones. The high usage and advancement of mobile banking applications to support the daily banking transactions has led to an insanely high rate of generation of data motivated by effortless expenditure. This suggests that complex analytical tasks are therefore needed for daily credit card data analysis and reporting.

2.1.3. Variety

The variety is mostly associated with the format in which data is stored (Nyikes & Rajnai 2015, pp. 217 – 222; Trelewicz 2017, pp. 1 - 3). Data can be stored in multiple traditional formats such as excel, databases, and comma-separated values (csv), among others. Data is not always stored in traditional formats; instead, it can also be stored in other formats such as video, PDF, SMS, MMS, email and others. However, in this research study, the same format is used for the credit card data that was collected. This is because data in the same format is easy to rearrange. It should however be noted that real-world data is not always in the same format. Complex analytical tasks are required for the analysis of data stored in an unstructured format.

2.1.4. Veracity

The veracity is mostly associated with the trustworthiness of the data (Nyikes & Rajnai 2015, pp. 217 - 222). The quality and the usability of data is highly dependent on the source system. Therefore, not all data is good for analysis. The accumulated data that is mined to uncover the meanings and unknowns in the data must be meaningful to the problem to be analysed.

In consideration of the four V's, big data can be defined as a complex and fast-growing data composed of different data formats that cannot be processed by traditional data management applications to maintain the security and the integrity of data. In the section that follows, credit card fraudulent activities that negatively affect the banking industry are discussed.

2.2. Credit Card Fraud Trends

When credit card holders make online or offline payments, fraudsters use sophisticated methods to capture credit card details of card holders that are ignorant of suspicious activities. Fraudsters take advantage of the ignorance of the card holder by making as many clean transactions as possible. Fraudsters can either commit online or offline credit card fraud.

2.2.1. Online Credit Card Fraud

Online credit card fraud regarded as Card Not Present (CNP) fraud whereby credit card details are stolen without stealing the physical credit card (Hsu & Chao 2007, pp. 1 – 4; Babu & Rajeshwari 2016, pp. 439 - 444). The information in the stolen credit card such as card number, expiry date and card verification value (CVV) allows the card to be used even when it is not physically present (e.g. in internet purchases).

Online credit card fraud such as CNP continues to grow globally due to advanced data analytics strategies used by credit card fraudsters. In this study, it is argued that fraudsters are innovative experts using sophisticated strategies that utilize weaknesses of standard machine learning strategies. In a nutshell, fraudsters take advantage of weak classifier models. To deploy a successful online credit card fraud solution, a technique that overcomes the weaknesses of classifier models is required.

The other types of online credit card frauds are as follows.

- **Site Cloning**

  Site cloning arises when a legal and official website is cloned by fraudsters to mislead clients into placing an order with them (Babu & Rajeshwari 2016, pp. 439 – 444; Rajak & Mathai 2015, pp. 1 - 4). The cloned website looks similar to legal and official website, and unaware clients would normally enter their credit card details on the cloned website to complete the online purchase. Upon receipt of the credit card details of the client, the fraudsters can make unauthorised fraudulent purchases until the customers and/or the bank becomes aware of the fraudulent credit card activities.

- **False Merchant Sites**

  This type of fraud is committed through websites that require customers to confirm their personal information by using credit card information (Babu & Rajeshwari 2016, pp. 439 – 444; Rajak & Mathai 2015, pp. 1 - 4). Such website owners can commit credit card fraud by themselves or by selling information of credit card-holders to fraudsters who will then commit the credit card fraud.

- **Phishing**

  This is the fraud through which fraudsters send out misleading e-mails to users in order to gain access to their personal information (Babu & Rajeshwari 2016, pp. 439 - 444). Fraudsters use very intelligent and legitimate methods to convince users into believing these e-mails. Example of such methods include providing users with links to websites asking these users to enter their personal information.

- **Friendly Fraud**

  This is when the fraudster is the credit cardholder that uses the card for offline purchases at actual stores or online purchases and later report the card as stolen or lost in order to claim for reimbursement from the bank (Babu & Rajeshwari 2016, pp. 439 - 444).

A successful implementation of the proposed model that overcomes the weaknesses of weak classifier models will reduce the level of online credit card fraud is committed. The stacking ensemble method, which combines two or more classification models, is used to vote an output of each transaction. The vote of an output is based on weights of classification model per transaction. This means that a classification model that predicts an output of a transaction badly would have a vote with lesser weight for predicting the final output of a transaction. Therefore, weak classifier models that fraudsters take advantage of would not contribute much in the prediction of the final output of a transaction. The next subsection provides a detailed discussion of offline credit card fraud.

2.2.2. Offline Credit Card Fraud

Offline credit card fraud is the type of fraud through which a physical card is stolen to make purchases at the actual physical stores (Babu & Rajeshwari 2016, pp. 439 – 444; Rajak & Mathai 2015, pp. 1 - 4). Offline credit card fraud is less common since there is a higher possibility of failure, and fraudsters normally prefer clean moves where stolen credit card is used to commit fraud without the fraudster being caught. In a case of credit card being stolen and not reported, some banks hold credit cardholders responsible for the losses suffered should illegal transactions occur.

An advanced solution for offline credit card fraud could be the usage of biometric systems for authorizations of credit card transactions. However, the biometric solutions are outside the scope of this study. In this study, the deployment of a successful offline credit card fraud solution is directed towards addressing a need for a technique that overcomes the weaknesses of classifier models. Classes of offline credit card are as follows.

- **Card Skimming**

  Fraudsters normally use special skimmer devices to capture credit card information stored in the magnetic stripes of the physical credit card (Hsu & Chao 2007, pp. 1 – 4; Babu & Rajeshwari 2016, pp. 439 - 444). Fraudsters can use captured credit card information to make duplicate copies of physical credit cards in order to make purchases either offline at the actual stores or online.

  At the ATM machine, fraudsters can place skimmer devices and micro-cameras to record the entered card pins when transaction are performed (Hoffman 2017). Card skimming can also be performed by corrupt employees that swipe customers' cards on skimmer devices during payments at merchant stores (Writer 2015).

- **Credit Card Generators**

  This is a type of fraud through which automated programs that use bank algorithms to generate credit card numbers arbitrarily and apply try and error techniques and other methods to research generated credit card numbers that

correspond to real credit card accounts from the actual banks (Rajak & Mathai 2015, pp. 1 - 4).

Since there is a high likelihood of failure for offline credit card fraud, a successful implementation of the proposed model that overcomes the weaknesses of weak classifier models will discourage fraudsters from attempting to perform offline credit card fraud. Thus, the rate at which online credit card fraudulent activities are performed will reduce. A stacking ensemble method that combines two or more classification models to vote an output of each transaction. The vote of an output is based on weights of classification model per transaction. On that premise, a classification model that predicts an output of a transaction badly would have a vote with lesser weight to predict the final output of a transaction. As a result, weak classifier models that fraudsters take advantage of would not significantly contribute to the prediction of the final output of a transaction. In the next sub-section, a detailed discussion on credit card application fraud is presented.

2.2.3. Credit Card Application Fraud

Credit card application fraud refers to when false information is used by fraudsters to apply for a credit card at the bank. The application fraud of credit card is associated with identity theft and chain of trust fraud. This type of fraud occurs very rarely since fraudulent behaviour can be detected during the application process when checking or verifying personal information.

An advanced solution for credit card application fraud could be the use of biometric systems to verify the personal information through genetic information such as finger prints, voice pattern, and eyes pattern. However, biometric solutions are outside the scope of this study. In this regards, data analytics methods for anomaly detection can only be used to prevent the continuity of credit card application fraud. Below is the discussion of offline credit card fraud, which can be eliminated by successful implementation of data analytics methods for anomaly detection. An aspect of credit card fraud that can be eliminated by successful implementation of data analytics for anomaly detection is identity theft, which is defined as follows:

- **Identity Theft**

> Identity theft involves fraudsters obtaining personal information such as, but not limited to, identity number, gender, e-mails and address from other people in order to open new accounts or to access existing accounts with the aim of committing fraud (Babu & Rajeshwari 2016, pp. 439 - 444). This is the type of fraud through which a fraudster commits credit card application fraud by using an identity that is not real or by stealing other people's identity with intensions of committing secondary fraud (Hsu & Chao 2007, p. 1 – 4; Babu & Rajeshwari 2016, pp. 439 - 444).

A fraudster that uses false identity and manages to get the credit card from the bank would normally use the credit card and later not pay the bill at a great loss to the bank. In a case whereby a fraudster uses the identity details of a real person and thereafter manages to get a credit card from the bank, the real person is liable for the payment of the bill unless the real person is able to provide proof of identity theft to the bank.

Since application fraud behaviour can be detected during the application process when checking or verifying personal information, the chances of failure are high. The successful implementation of the stacking ensemble method will discourage fraudsters from attempting to commit credit card application fraud.

2.3. Information and Data Security

2.3.1. Data Protection

Section 14 of the constitution of South Africa (1996) is designed to protect the privacy of the citizens of South Africa through limited rights to privacy. The Protection of Personal Information Act (POPI) Act was enacted to regulate access and use of personal information. The eight POPI principles are defined to govern lawful use of personal information (Government 2013).

2.3.2. Access to Data

Public sector agencies in South Africa normally have access to important and useful information about citizens (Chase 2018). The personal data from trusted public sector agencies can be used by South African banks to verify information of applicants for loans. Section 14 of the constitution of South Africa and the POPI Act limits banks

from having complete access to individual's personal data, which on the other hand motivates a rapid growth of identity theft related fraud (Eurofinas & ACCIS 2011).

2.3.3. Consent and Disclosure of Sensitive Information

When an applicant applies for a credit card at the bank, an applicant's provided information must be verified by the bank. For verification of applicant's information, banks must define a clear and a transparent privacy policy for handling sensitive information, and a consent in writing that shows the data subject that specify the intended usage of the data. The collection of sensitive information from public or private sectors (that of the same level of data protection as the bank under the privacy rules) must be lawful, it must be collected for lawful purpose, and it must be collected for only the intended purpose for which it has been collected for (Ahluwalia & Mahajan 2011).

When banks require sensitive information about credit card applicants, a lawful contract that permits disclosure of sensitive information must be present before disclosing of any sensitive information by private or public sector to any bank, unless such disclosure is required by the law (Ahluwalia & Mahajan 2011).

2.3.4. Cross Border Data Transfer

Sensitive information of international applicants in South Africa can be verified by South African banks. According to (Ahluwalia & Mahajan 2011), banks normally transfer sensitive information across the border if the entity to receive the information ensures the following:

- The same level of data protection as the bank under the privacy rules.
- The transfer is under a lawful contract between bank and the data subject.
- A person has signed a consent that gives an authority to transfer their sensitive information across the border.

An advanced analytics strategy for complex data analysis and verification of information of the credit card applicants in banks include machine learning, which is discussed in the next section.

2.4. Machine Learning Historical Foundation

Previously, computers were primarily used for searching information in large databases (Woodford 2017) while complex mathematical calculations were performed using calculators (Ball & Flamm 1996). The computer storage and capabilities were increased to accommodate an increase in the size of industrial data (Woodford 2017). For decision making, the increase in data needed to be analysed to support industrial processes.

The tools used for data analysis were used for regular tasks (Woodford 2017) and not for automatic classification of information or other machine intelligence tasks. The introduction of machine intelligence algorithms that can make decisions without intervention of humans became important for the management of complex computation. Hence, machine learning became a strategy for allowing machines to learn and act from data without being explicitly programmed (Andrew 2017).

Knowledge discovery and classification and prediction are two fundamental areas of machine intelligence (Zaza & Al-Emran 2015, pp. 275 - 279). A decision support system caters for possible decisions based on the patterns that are found in data. To this end, data mining is the machine intelligence that identifies the important patterns for prediction. One of the types of learning for data models is supervised learning; and it is discussed below.

2.4.1. Supervised Learning

This is the machine learning type in which the learning of models is guided by the class values (Brownlee 2016). The class values are either 0 for legal transactions or 1 for illegal transactions. A supervised machine learning data model must be supplied with credit card historical data that it must learn from in order to be in a good position to predict the outcomes of future transactions. The prediction labels of future transactions are done through the classification process by using machine learning classification algorithms. The classification is discussed in the next section.

2.5. Classification

This is one of the most widely used functionality category methods in machine learning. The application of classification methods includes fraud detection analysis,

medical diagnosis analysis, spam detection analysis, and risk assessment analysis. The main goal of classification methods is to classify an output (y) into a category given some values of x (Peng, et al. 2017, pp. 1 – 7; Cui, et al. 2018, pp. 1 - 11). That is to say, based on prediction of attributes for prediction, a classification method assigns variable to the target category. For the prediction of attributes, the classification method uses some statistical, mathematical and computational algorithms such as, but not limited to, Artificial Neural Network, Regression, Nearest Neighbour, Decision Trees, and Support Vector Machine (Cui, et al. 2018, pp. 1 – 11; Liu, et al. 2018, pp. 1 - 13).

This research study involves classification, whereby data analytics algorithms are trained to separate credit card fraudulent transactions from non-fraudulent transactions. Fraudulent transactions are grouped together as one class by labelling predicted output as 1, and non-fraudulent transaction are grouped together as another class by labelling predicted output as 0. In the next sub-section, the types of data analytics are discussed.

2.6. Types of Data Analytics

2.6.1. Descriptive Analytics (What's happening?)

This is an analytic that provides insight into the data by describing the past (Gomes, et al. 2016, pp. 1 – 5; Haneem, et al. 2017, pp. 1 - 6). The past in this case may refer to an occurrence of an event that is one minute ago, a week ago, or a year ago. Descriptive data analysis is important and useful for enabling people to understand and learn from past behaviours in order to understand how future outcomes might be influenced.

Mathematical and statistical methods such as, but not limited to, averages, sums, and percentages are normally used for performing aggregations of filtered columns of data to generate information such as year over year changes in an organization and total stock, average money spent by each customer. Common examples of descriptive analytics provide historical insight on the company's data such as production, operations, customers, sales, and financial figures.

2.6.2. Predictive Analytics (What is likely to happen?)

Predictive analytics are statistical and machine learning algorithms that dig into the historical data obtained from systems such as customer relation management (CRM), enterprise resource planning (ERP), human resource (HR), re-organise the data and categorize the data to perform the forecasting of the insights about the future based on the collected data (Queiroz, et al. 2016, pp. 1 – 6; Toporek, et al. 2011, pp. 1 – 4; Li, et al. 2017, pp. 1 - 6). Predictive analytics provide the financial industry with actionable insights based on collected data.

Predictive analytics can be used in the banking industry to forecast purchasing patterns, customer behaviours, and to produce credit score. Predictive analytics is based on probabilities to estimate the likelihood of future outcomes. Probability is the subset of statistics; hence, statistical algorithms cannot predict the future outcomes with 100% certainty. Predictive analytics is, therefore, an analytical method that answers the question of what is likely to happen in the future. Challenges associated with detection techniques are discussed in the next section.

2.7. Stacking Algorithm

There is no machine learning method that performs well for all measures in every given application according to the NO FREE LUNCH (NFL) theory (Macready & Wolpert 1996, pp. 67 - 82). Stacking is an ensemble method that uses various machine learning algorithms to construct a model that is used to generate new dataset that is used in a combiner machine learning algorithm (Smolyakov 2017). In the context of classification stacking models, different classifier models are trained with the dataset and produce an output that is either 0 or 1.

The output column (dependent variable) of base classifier models are used as features of the stacking model. The stacking ensemble technique is used to generate the output value based on the voting method. For each data observation, the stacking model uses either the majority voting method or the weighted voting method for the prediction of the final output value. Majority voting refers to when all base models vote a prediction class of each transaction. On the other hand, weighted voting involves the voting carried out by counting the predictions of better models to predict the output of

each transaction. In this research, the weighted voting has been adopted for use in the stacking ensemble model for the classification of the output of each transaction.

2.7.1. Weighted Voting Method

It is assumed $\delta_1, \delta_2, \delta_3, \dots, \delta_n$ is be the weight of transaction, and $\beta_1, \beta_2, \beta_3, \dots, \beta_n$ is the output values of transactions, where β is the output of transaction, δ is the weight of transaction, and $1,2,3, \dots, n$ is the transaction number from 1 until transaction n. The term “non-fraudulent” is the opposite of fraudulent. Mathematically, the term “opposite” can be represented by the negation. If the fraudulent transaction is represented by 1, the non-fraudulent transaction can therefore be represented by -1.

For weighted voting, the study replaces the output 0 of base models with -1 since direction is important. If the output value of Weight_Voting is positive then the transaction is fraudulent, and if the output value of Weight_Voting is negative then the transaction is non-fraudulent. The formula for Weight_Voting is defined as follows:

$$\text{Weight\_Voting} = \beta_1\delta_1 + \beta_2\delta_2 + \beta_3\delta_3 + \dots + \beta_n\delta_n \qquad (2.1)$$

Furthermore, in this study, it is proposed that weighted voting for the classification models require an optimization of weights to produce good results. Thus, differential evolution algorithm is a stochastic method to be used for optimization of the weights of transactions in this research study. Differential evolution method is discussed in the next sub-section.

2.8. Differential Evolution Algorithm

This is an optimization technique of stochastic search for a population-based vector, and it is powerful and efficient over a continuous space for solving differentiable and non-linear problems. This algorithm was introduced in 1996 by Storn and Price. The algorithm is an evolutionary method composed of five sequential stages, namely: population initialization, mutation, recombination, selection, and termination criteria. If the data observation does not meet the termination criteria, the data observation evolves to the next generation where the weight is readjusted. If the data observation meets the termination criteria, the data observation is an optimal solution.

Consider the optimization problem f(x) where X = $[x_1, x_2, x_3, \dots, x_D]$, and D is the number of variables. For the population size N and a generation G, the population matrix is defined as $x_{n,i}^G = x_{n,1}^G, x_{n,2}^G, x_{n,3}^G, \dots, x_{n,D}^G$.

The five sequential stages of the evolutionary method are discussed in the following sub-sections.

2.8.1. Initial Population

The random generation of the initial population between lower bound and upper bound is defined by the following equation:

$$x_{n,i} = x_{n,i}^L + rand() * \left(x_{n,i}^U - x_{n,i}^L\right), i = 1,2,3, \dots, D \text{ and } n = 1,2,3, \dots, N \qquad (2.2)$$

where x_i^L is the lower bound of the variable x_i, and x_i^U is the upper bound of the variable x_i.

2.8.2. Mutation

For each parameter vector, three other vectors $x_{r1n}^G, x_{r2n}^G \text{ and } x_{r3n}^G$ are selected randomly and the weighted difference of two of the vectors is added to the third

$$x_n^{G+1} = x_{r1n}^G + F(x_{r2n}^G - x_{r3n}^G) \qquad (2.3)$$

where n = 1,2,3, …, N. x_n^{G+1} is called a donor vector and F is a user supplied value that is taken between 0 and 1.

2.8.3. Recombination

The trial vector $U_{n,i}^{G+1}$ is derived from the donor vector $V_{n,i}^{G+1}$ and the target vector $x_{n,i}^G$ in accordance with the following equation.

$$U_{n,i}^{G+1} = \begin{cases} U_{n,i}^{G+1} & if\ rand(\ ) \le Cp \text{ or } i = Irand \\ x_{n,i}^G & if\ rand(\ ) > Cp \text{ and } i = Irand \end{cases} \qquad (2.4)$$

$$i = 1,2,3, \dots, D \text{ and } n = 1,2,3, \dots, N$$

where $Irand$ is an integer random number between I and D, and Cp is the recombination probability that is chosen randomly between the value of 0 and 1.

2.8.4. Selection

The trial vector $U_{n,i}^{G+1}$ in the equation 2.11 is compared with the target vector $x_{n,i}^{G}$, and the one with the lowest function value is selected for next generation.

$$x_n^{G+1} = \begin{cases} U_{n,i}^{G+1} & if\ f(U_n^{G+1}) > f(x_n^G) \\ x_n^G & Otherwise \end{cases} \quad (2.5)$$

$$n = 1,2,3, \dots, N$$

The differential evolution method is a stochastic method that is used to optimize the weights of voting the output of transactions. The next sub-section discusses challenges associated with detection techniques.

2.9. Challenges Associated with Detection Techniques

2.9.1. Skewed Distribution

The skewed distribution is also known as the unbalanced class distribution (Godase & Attar 2012, pp. 1 – 6; Somasundaram & Reddy 2017, pp. 1 – 6; Kho & Vea 2017, pp. 1 - 5). Therefore, in the event of a need to build a classification supervised learning model to classify new samples as to whether they belong to class A or class B, the labelled training data observations is required. The skewed distribution is a result of many labelled training samples in class B than in class A or vice-versa (Godase & Attar 2012, pp. 1 – 6; Somasundaram & Reddy 2017, pp. 1 - 6).

In a skewed class distribution, the standard machine learning algorithms that achieve high accuracy tend to classify all credit card transaction as the class with the majority data observations which is always the non-fraudulent class (Zeager, et al. 2017, pp. 112 - 116). The classification of all transactions as non-fraudulent implies poor predictive accuracy of fraudulent transactions. The skewed distribution causes the model to know less about the class with less labelled training samples and to know more about the class with many labelled training samples. The issue of skewed distribution is common in the context of credit card fraud detection since the number of legal observations are greater than the number of illegal observations (Pushpa & Malini 2017, pp. 255 - 258). Therefore, fraud detection model developers have to design ways to overcome the challenge when constructing a fraud detection system.

One possible solution to consider for balancing the class distribution is under-sampling and SMOTE oversampling. From the issue of unbalanced data in credit card transactions, it is realistic to assume that the many transactions from the class of the majority data observations are redundant. Therefore, random elimination of the redundant transactions would not change the structure of the dataset significantly. However, random removal of redundant transactions has risks, since the removal of redundant transactions is not done in an unsupervised manner.

In practice, sampling is the method of reducing the class of the majority data observations using the subset of the class to represent the entire population of the class. Several binary classifiers have shown better performance when they are trained with balanced class distribution (Gosain & Sardana 2017, pp. 79 - 85). However, in this research study it is not implied that the classifier models cannot learn and predict output from unbalanced class distribution. Zeager, et al. (2017, pp. 112 - 116) have shown that the use of sampling techniques for balancing the class distribution does not improve performance of some of their classifiers. Thus, in order to establish if sampling techniques enhance the performance of the classification models, some simulations have to be run.

The simulations must also include different ratios of class distribution to establish the optimum ratio. In a highly skewed class distribution, the SMOTE oversampling method of the minority class to have a 50:50 ratio against the minority class improves the intersection between the two classes. Having a model with high percentage of accuracy does not necessarily mean the model is doing well to solve the problem.

In a case where there are 100 transactions, and 95 of those transactions are legal and 5 transactions are illegal, and the model accuracy is 95%. If the classification model predicts all 100 transactions to be legitimate, the model has a higher accuracy percentage yet the model is substandard. The model is regarded as being of poor quality because it is only knowledgeable about a single class.

However, if 90 transactions are classified as legitimate when 80 transactions are in fact legitimate, and 10 transactions are classified as fraudulent when 4 transactions are in fact fraudulent, the accuracy percentage in this case is 80%. The model is good because both classes are equally or nearly equally represented. To this end, the true accuracy of the model can be achieved by calculating the F-score or confusion matrix.

The true accuracy of the classifier is achieved by having the F-score or confusion matrix values that represent both classes.

The poor performance of the classification model is not only caused by a smaller number of illegal transactions in the class of the minority data observation in relation to higher number of legal transactions in the class of the majority data observation, but also by the overlapping of between two classes of transactions. Additionally, the performance of the transaction classifier in unbalanced binary data distribution is affected by the availability of noise in transactions. The noise is discussed in the next subsection.

2.9.2. Noise

Noise refers to the possible errors that are associated with data (Lita, et al. 2007, pp. 1 – 5; Somasundaram & Reddy 2017, pp. 1 – 6; Li, et al. 2018, pp. 1 - 10). Examples of errors that might be found in data include missing values and incorrect dates. An error in data can result in an erroneous model construction (Somasundaram & Reddy 2017, pp. 1 – 6; Li, et al. 2018, pp. 1 - 10), and a model with errors can result in prediction accuracy of poor quality. Another challenge associated with detection techniques, which is discussed in the next section, is overlapping data.

2.9.3. Overlapping Data

Overlapping data poses a challenge to the supervised learning methods. The challenge of overlapping data points is as a result of a legal transaction looking nearly the same as the illegal transaction or vice versa (Somasundaram & Reddy 2017, pp. 1 – 6; Mak, et al. 2011, pp. 1 - 6). This is a serious problem since an erroneous model constructed will detect transactions as fraudulent when they are in fact not fraudulent or allow fraudulent transactions to go undetected since they are mistakenly treated as legal transactions (Zhou, et al. 2014, pp. 1 - 6). The challenge of detection techniques is choosing parameters is discussed in the following sub-section.

2.9.4. Choosing Parameters

Most of machine learning algorithms require several parameters and thresholds values defined to pre-set by the users (Silva & Wunsch 2017, pp. 1 – 8; Basha, et al. 2017, pp. 1 - 6). Different parameters in machine learning algorithms normally produce different results (Silva & Wunsch 2017, pp. 1 - 8). Hence, various parameters can lead

to a model with totally changed performance. A model with completely different performance will increase the complexity of model construction. The challenge of detection techniques that is in line for a discussion is feature selection.

2.9.5. Feature Selection

Feature selection is the method of picking the subset of features relevant to the problem from the total number of given features related to the problem domain (Singh, et al. 2015, pp. 388 – 393; Todd & Harvey 2015, pp. 474 – 489; Kamal, et al. 2009, pp. 1 - 6). For all features excluding a labelled feature, a subset of features is achieved by eliminating one of two related features that shows a strong correlation because they have the same impact on the data model. Thus, elimination of one of two related features reduces the complexity of the data model while preserving the variety of features. Given a variety in the features of the dataset, it is not always necessary to perform feature reduction because the data model might lose useful information. Knowing which relevant features to use when constructing a predictive model is a challenge that needs deep understanding of the problem domain (Chezian & Devi. 2016, pp. 161 – 165; West & Bhattacharya 2016, pp. 1734 – 1744; Mei & Jiang 2016, pp. 301 - 305).

Feature Engineering is a necessary step of data preparation to reduce computational time and complexity and to avoid overfitting of the data model (Drotár & Gazda 2016, pp. 1 – 5; Qiu, et al. 2018, pp. 1 – 4; Tran & Li 2009, pp. 1 – 4). The feature engineering methods discussed in this research study are Univariate Feature Selection Method, Recursive Feature Elimination Method and Feature Importance. The first feature engineering method to discuss is Univariate Feature Selection Method and it is discussed in the subsection below.

- **Univariate Feature Selection Method**

  This method seeks to analyse the relationship between each feature of the input dataset and the output feature of the dataset. Univariate statistical test is conducted to test whether there is any statistically significant relationship between each input feature of the dataset. The Univariate Feature Selection Method was implemented successfully by (Drotár & Gazda 2016, pp. 1 – 5; Curtis & Kon 2015, pp. 1 – 4; Subho, et al. 2019, 1 – 6). The next feature engineering method to discuss is Recursive Feature Elimination Method.

- **Recursive Feature Elimination Method**

  This model select features of the input dataset by recursively selecting smaller subset of the input features. The initial subset of the input features was used to train the estimator. The estimator is used to estimate the feature importance. The least important features are pruned from the subset of the input features. The Recursive Feature Elimination Method was implemented successfully by (Tran & Li 2009, pp. 1 – 4; Lv, et al. 2014, pp. 1 – 4; Zeng, et al. 2009, pp. 1 - 4). The next feature engineering method to discuss is Feature Importance.

- **Feature Importance Method**

  This method assigns scores to the input features of the dataset based on how useful the input features are classifying the output feature. The feature importance scores are obtained by computing the correlation between each input feature and the output feature. The feature important scores can provide an insight into the dataset, model and the classification model. The Feature Importance Method was implemented successfully by (Chu, et al. 2019, pp. 1 – 3; Qiu, et al. 2018, pp. 1 – 4; Dutta, et al. 2018, 1 – 5)

This section highlighted some of the practical issues when solving a machine learning problem with a skewed class distribution. Generally, classification data models are not designed to work well with skewed data distribution. Therefore, various data pre-processing actions are taken to clean and shape datasets such that the classification process of credit card data can be adequately performed to produce high predictive accuracy of both fraudulent and non-fraudulent transactions without introducing any biasness or high variance in the data model. Works of other researchers that are related to this research study are discussed in the next section.

2.10. State-of-the-Art and Background Theory

In this section, the theoretical background of the state-of-the-art detection techniques for credit card fraud are discussed. Well analysed data produce good results that enhance the quality of decision making by organizations. The daily losses suffered by banks which result from credit card fraud indicate lack of complex analytical tasks to prevent fraudulent activities related to credit card. The state-of-the-art models are discussed below.

2.10.1. Logistic Regression (LR)

The LR in this research study is focused on the binary classification of transactions where the predicted value y can only take the value of 1 for the class of fraudulent transactions or the value of 0 for class non-fraudulent transactions. Thus, y∈{0,1}. The corresponding $y^{(i)}$ of $x^{(i)}$ is called the label for the training sample (Liu, et al. 2017, pp. 1 – 6; Pavlyshenko 2016, pp. 1 – 5; Ding, et al. 2017, pp. 1 - 6). Hypothesis for logistic regression is logistic function or sigmoid function (Pushpa & Malini 2017, pp. 255 – 258; Andropov, et al. 2017, pp. 1 – 6; Srivastava, et al. 2016, pp. 1 – 4; Liu, et al. 2017, pp. 1 – 6; Pavlyshenko 2016, pp. 1 - 5); and it is defined as follows: $h_\theta(x) = g(z)$, where z = $\theta^T x$. Thus, $g(z) = \frac{1}{1+e^{-z}}$.

A pictorial representation of LR is captured in figure 2.1.

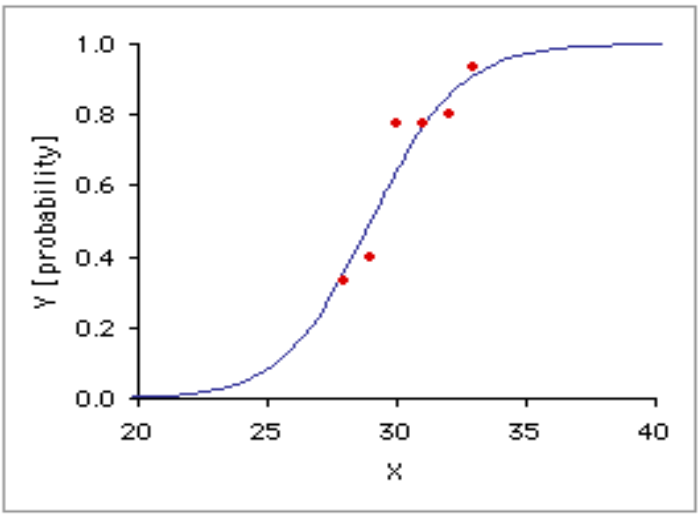


Figure 2.1: A depiction of the Logistic Regression.

The sigmoid function is used to map any real number to the interval between 0 and 1. Thus, the function $h_\theta(x)$ must satisfy $0 \leq h_\theta(x) \leq 1$. The function will give out the probability that an output is 0 or 1 (Y.Liu, et al., 2017; Ding, et al., 2017). Any probability greater or equal to 0.50 suggests an output of 1 ($h_\theta(x) \geq 0.5 \rightarrow y = 1$), and any probability that is less than 0.50 suggests an output of 0 ($h_\theta(x) < 0.5 \rightarrow y = 0$). The cost function for logistic regression is defined as: $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(h_\theta(x^{(i)}), y^{(i)})$

where $cost(h_\theta(x^{(i)}), y^{(i)})$ = -log ($h_\theta(x)$) if the output value y = 1 or $cost(h_\theta(x^{(i)}), y^{(i)})$ = -log ($1 - h_\theta(x)$) if the output value y = 0. The simplified cost function that accommodates both values of y = 1 and y = 0 is defined as follows: $cost(h_\theta(x^{(i)}), y^{(i)})$

= -(y) log ($h_\theta(x)$) – (1-y) log (1-$h_\theta(x)$). Notice that both (y) and (1-y) are added to the equation such that if y = 0 then:

$cost(h_\theta(x^{(i)}), y^{(i)})$ = -(y) log ($h_\theta(x)$) – (1-y) log (1-$h_\theta(x)$)

= -(0) log ($h_\theta(x)$) – (1-0) log (1-$h_\theta(x)$)

= -log (1 – $h_\theta(x)$)

And if y = 1 then: $cost(h_\theta(x^{(i)}), y^{(i)})$ = -(y) log ($h_\theta(x)$) – (1-y) log (1-$h_\theta(x)$)

= -(1) log ($h_\theta(x)$) – (1-1) log (1-$h_\theta(x)$)

= -log ($h_\theta(x)$)

Thus, the addition of (y) and (1-y) to the simplified cost function equation of logistic regression would not affect the results. The simplified cost function of logistic regression is therefore defined as:

$$J(\theta) = \frac{-1}{m}\sum_{i=1}^{m}[(y^{(i)})\log\left(h_\theta(x^{(i)})\right) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]. \quad (2.6)$$

The logistic regression in this study is used as one of the base models of stacking ensemble method. The logistic regression model learns the training dataset. The next subsection discusses the support vector machine model.

2.10.2. Support Vector Machine (SVM)

The logic behind SVM is derived from logistic regression and sigmoid function to classify the output as either 1 or 0. To explain the support vector machine cost function, the logistic regression cost function formula and sigmoid function formula, which are explained in detailed from logistic regression section, are tackled first (Pushpa & Malini 2017, pp. 255 – 258; Andropov, et al. 2017, pp. 1 – 6; Srivastava, et al. 2016, pp. 1 – 4; Liu, et al. 2017, pp. 1 – 6; Pavlyshenko 2016, pp. 1 - 5). The sigmoid function is defined as follows: $h_\theta(x) = g(z)$, where z = $\theta^T x$. Thus, $g(z) = \frac{1}{1+e^{-z}}$.

SVM is illustrated pictorially if Figure 2.2.

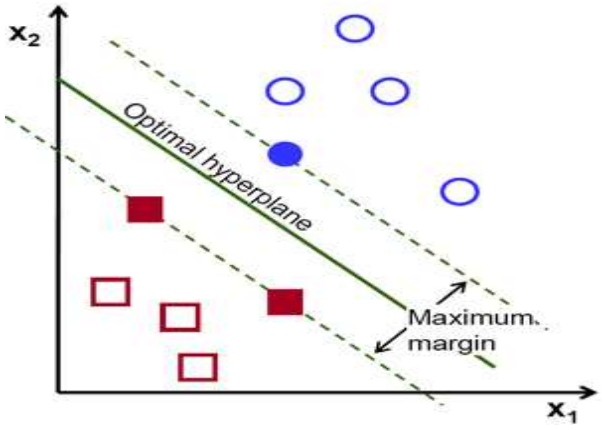


Figure 2.2: An illustration of Support Vector Machine.

The LR loss function is defined as follows:

$$J(\theta) = -[(y)\log\left(h_\theta(x)\right) + (1-y)\log(1-h_\theta(x)]$$

$$= -(y)\log\left(h_\theta(x)\right) - (1-y)\log(1-h_\theta(x)]$$

If the output value is y = 1, then the cost function is $-\log(h_\theta(x))$, the goal is for $\theta^T x$ to be much greater than 0 ($\theta^T x >> 0$). If the output value is y = 0, then the cost function is $-\log(1 - h_\theta(x))$, The goal for $\theta^T x$ to be much lesser than 0 ($\theta^T x << 0$). ($\theta^T x >> 0$) is the cost of z when y = 1 and it is denoted as $Cost_1(z)$, and ($\theta^T x << 0$) is the cost of z when y=0 and it is denoted by $Cost_0(z)$. From logistic regression, $-\log(h_\theta(x^i))$ will be replaced with $Cost_1(\theta^T x^{(i)})$ and $-\log(1 - h_\theta(x^{(i)}))$ with $Cost_0(\theta^T x^{(i)})$. If C = $\frac{1}{\lambda}$ and both m's cancel each other, then the simplified minimum cost function for support vector machine is:

$$J(\theta) = C\sum_{i=1}^{m}[y^{(i)}Cost_1(\theta^T x^{(i)}) + (1-y^{(i)})Cost_0(\theta^T x^{(i)})] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2 \qquad (2.7)$$

In this study, the support vector machine is used as one of the base models of stacking ensemble method. The support vector machine model will learn the training dataset. The artificial neural network model is discussed in the section that follows.

2.10.3. Artificial Neural Networks (ANN)

The ANN algorithm is designed to imitate the functionality of the human brain where the input values are called dendrites and the output values are called are called axons (Rout, et al. 2017, pp. 1 – 6; Miholca & Onicaş 2017, pp. 1 - 8). The dendrites represent the input features $x_1, x_2, x_3, \ldots, x_n$ of the problem (Basheer & Hajmeer 2000). The

hypothesis for ANN uses logistic function (Pushpa & Malini 2017, pp. 255 – 258; Andropov, et al. 2017, pp. 1 – 6; Srivastava, et al. 2016, pp. 1 – 4; Liu, et al. 2017, pp. 1 – 6; Pavlyshenko 2016, pp. 1 - 5); and it is defined as follows: $h_\theta(x) = g(z)$, where z = $\theta^T x$. Thus, $g(z) = \frac{1}{1+e^{-z}}$.

Sometimes the dendrites include the input node x_0, which is also known as bias unit, and is always equal to 1. The ANN hypothesis is composed of input layer, intermediate layer or layers also known as hidden layers, and the output layer (Andropov, et al. 2017, pp. 1 - 6). The hidden layers of the ANN nodes are denoted by $\alpha_i^{(j)}$, where "i" represents the unit number and "j" represent the layer number.

The regularised cost function is defined as follows:

$$J(\theta) = \frac{-1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[\left(y_k^{(i)}\right)\log\left(h_\theta\left(x^{(i)}\right)\right)_k + (1-y_k^{(i)})\log(1 - \log\left(h_\theta\left(x^{(i)}\right)\right)_k)] + \frac{-1}{m}\sum_{L}^{L-1}\sum_{i}^{S_L}\sum_{j}^{S_{L+1}}(\theta_{j,i}^{(l)})^2 \quad (2.8)$$

where K = is the number of axons (output).

L = is the total number of layers in the ANN; and

S_L = is the number of units in layer 1 excluding the bias unit (x_0).

The Figure 2.3 is pictorial representation of ANN.

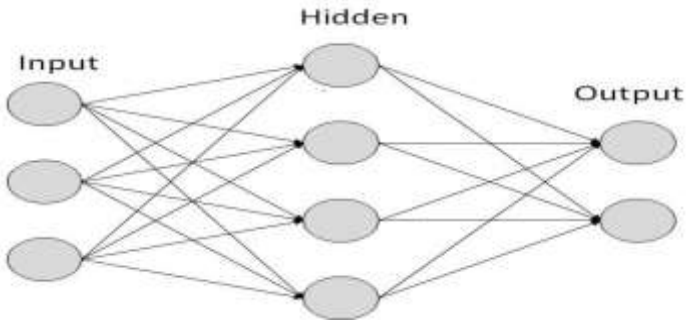


Figure 2.3: A depiction of Artificial Neural Network

The ANN cost function is actually the logistic regression cost function with the double summation to simply add up the cost calculated for each cell in the output layer. In this research study, the ANN is used as one of the base models of stacking ensemble method. The ANN model learns the training dataset. The k-nearest neighbour model is discussed in the following section.

2.10.4. K-Nearest Neighbour (KNN)

The KNN algorithm involves determining the relationship between x and y when given data with training sample (x, y) (Pushpa & Malini 2017, pp. 255 – 258; Vipani, et al. 2017, pp. 1 - 5). The aim is to learn the algorithm such that when given an unseen value of x the algorithm can confidently predict the output y value that corresponds to the x value. k-NN is essentially designed to classify the unseen value of x by calculating the distance of the nearest class (Pushpa & Malini 2017, pp. 255 – 258; Vipani, et al. 2017, pp. 1 - 5). Measures such as Manhattan, harming, Chebyshev, Euclidean distance are used to compute the distance. The most widely used measure is the Euclidean distance (Pushpa & Malini 2017, pp. 255 – 258; Vipani, et al. 2017, pp. 1 - 5). The following equation is a definition of the Euclidian distance function:

$$d(x,x') = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2 + \cdots + (x_n - x_n')^2} \qquad (2.9)$$

the k-NN classifier is performed using two steps (i.e. positive integer k and a similarity metric d) when given unseen value of x. The algorithm runs through the dataset capturing the similarity metric d between each training sample (x, y) and the input value of x. The k points of the training sample that are closer to input value x are stored in a separate container (e.g. set A). The conditional probability is predicted for each class with an indicator function $I(x)$, which predicts the output value y = 1 if the transaction is fraudulent or y = 0 if the transaction is not fraudulent. The conditional probability for each class with an indicator function $I(x)$ is defined as:

$$P(y = j \mid X = x) = \frac{1}{k} \sum_{i \in A} I(y^{(i)} = j) \qquad (2.10)$$

The KNN technique pictorially is illustrated in figure 2.4.

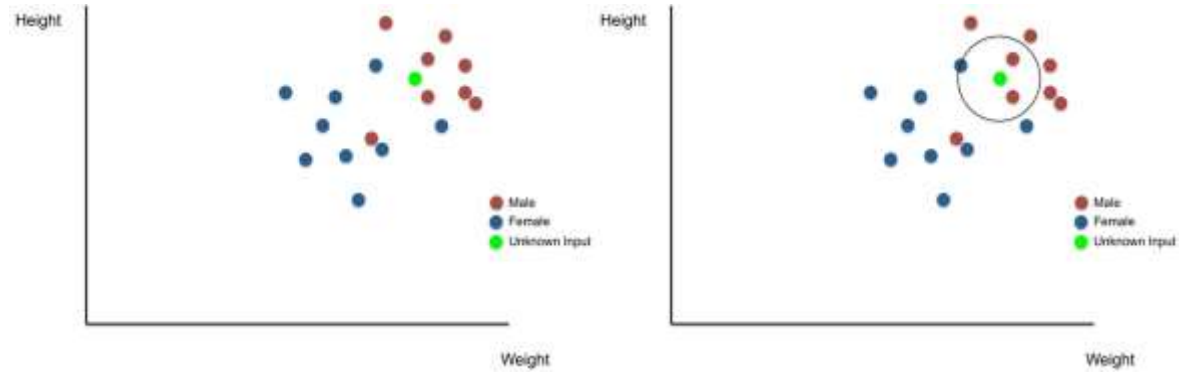


Figure 2.4: An illustration of the K-Nearest Neighbour technique.

In this study, the KNN is used as one of the base models of stacking ensemble method. The KNN model will learn the training dataset. The Naïve Bayesian model is discussed in the sub-section that follows.

2.10.5. Decision Tree (DT)

The DT is a data mining modelling technique for partitioning data into subsets based on the categories of input variables (Manjaramkar & Kokare 2017, pp. 1 – 4; Chaaya & Maalouf 2017, pp. 1 – 7; Zakerian, et al. 2017, pp. 1 - 6). This is a modelling technique presented in a tree-like structure which plays a crucial role that assists in the understanding of the path taken by decision makers when making an informed decision. After using historical data to learn the DT model, ways in which the model decides the class of the credit card transaction can be derived. The DT model is used as a classifier model to predict the output class of future credit card transactions. The Figure 2.5 is a pictorial representation of the DT model.

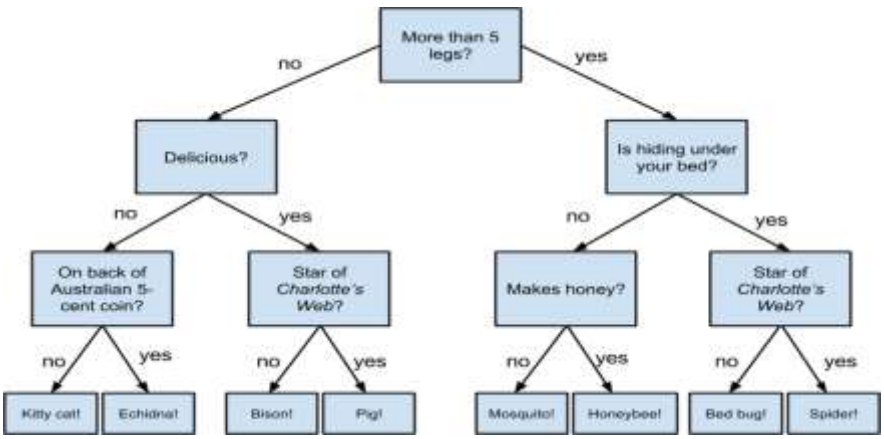


Figure 1.5: A depiction of the Decision Tree.

The DT model analyses the given data with the hope of finding the one variable that divides data into different logical groups (Manjaramkar & Kokare 2017, pp. 1 - 4). The decision tree models are widely used because they are easy to understand and interpret. DT model are known for handling missing values well (Chaaya & Maalouf 2017, pp. 1 – 7; Zakerian, et al. 2017, pp. 1- 6). Therefore, if a dataset contains missing values and people are interested in a quick and easily interpretable and understandable answer, it is important to use a DT. The next detection technique discussed is the Naïve Bayesian.

2.10.6. Naïve Bayesian (NB)

The NB classifier is based on the Bayes theorem (Han, et al. 2015, pp. 1 – 4; Katkar & Kulkarni 2013, pp. 1 - 6). Bayes theorem is named after Thomas Bayes and it is based on conditional probability (Meiping 2009, pp. 1 - 4). Conditional probability is the probability that predicts the outcome, given some conditions or events that have occurred (Han, et al. 2015, p. 1 – 4; Katkar & Kulkarni 2013, pp. 1 - 6). The function of the conditional probability as follows: $P(A\backslash B) = \frac{P(B\backslash A)\times P(A)}{P(B)}$, where

P(A) is the likelihood of the hypothesis A being correct; P(B) is the likelihood of events that have occurred; P(B\A) is the likelihood of the hypothesis given an event that has occurred; and P(A\B) is the likelihood of the hypothesis given an event that has occurred. The Naïve Bayesian technique is depicted if Figure 2.6.

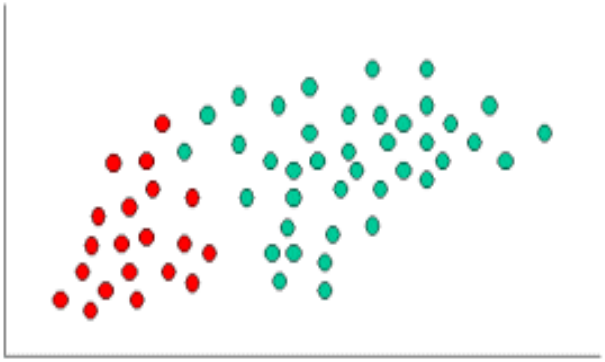


Figure 2.2: A depiction of the Naïve Bayesian technique.

Since the naïve Bayes classifier uses the Bayes theorem, the classification model predicts the likelihood of an input data record belonging to a particular class. The input records are likely to belong to a class with the highest likelihood value. The naïve

Bayesian classifier is used as one of the base models of stacking ensemble method. The naïve Bayesian classifier model will learn the training dataset.

In this research study, the stacking ensemble method uses the various state-of-the-art algorithms for covering a variety of credit card transaction. The stacking ensemble method improves the predictive capabilities of the data model by voting the class of each transaction using the state-of-the-art techniques. The stacking ensemble technique is used to generate new dataset from model that poorly predict the output of credit card transactions. The weights of new generated dataset are adjusted to improve the predictive capability of the model. The next section provides details works related to fraud prevention technologies.

2.11. Related Works in Fraud Prevention Technologies

Credit card fraud is a criminal activity that financially benefits an individual or group of individuals. It is deliberately carried out by individuals working against the law. Fraud prevention technologies have been used in the banking industry for a long time to prevent fraudulent transactions. However, since fraud masters are adaptive, they normally find a way around credit card prevention technologies.

Credit card fraud detection within the banking industry is an evolving discipline for detecting non-preventable fraudulent transactions. Machine learning and statistics are two broad fields that have demonstrated their effectiveness in fraud detection. Common state-of-the-art statistical and machine learning techniques for classification problems utilized for the fraud detection of the credit card transactions include techniques such as LR, ANN, SVM, decision tree, k-NN and outlier detection (Malini & Pushpa 2017, pp. 1 – 4; Ganji 2012, pp. 1 – 5; Das, et al. 2017, pp. 1 – 4; Manjaramkar & Kokare 2017, pp. 1 – 4; Liu, et al. 2017, pp. 1 – 6; Mao, et al. 2017, pp. 1 - 8).

Given a dataset, some classification techniques are more successful at detecting future credit card fraudulent transactions than others. These classification techniques need historical data for the effective prevention of future credit card transactions. However, historical data is usually accompanied by challenges associated with imbalanced data, whereby the percentage of illegal transactions is far lower than the

percentage of legal transactions (Gosain & Sardana 2017, pp. 79 – 85; Matsuda & Murase 2016, pp. 349 – 354; Pengfei, et al. 2014, pp. 217 - 222).

To address the issue of imbalanced data, techniques such as oversampling of the class of minority data observations and under-sampling of the class of majority data observations are used. In this research study, oversampling techniques such as safe-level Synthetic Minority Oversampling Technique (SL-SMOTE) and Synthetic Minority Oversampling Technique (SMOTE) are used to generate synthetic data observations between the two data observations of the minority class. SMOTE and SL-SMOTE are sampling methods which oversample the minority class by computing median features vectors between nominal features sample and its potential nearest neighbours through Euclidean distance of standard deviations (Chawla, et al. 2002, pp. 321 - 357). However, SL-SMOTE generates synthetic data observations between the two data observations at the safe level of the minority class. The use of SL-SMOTE and SMOTE have been successful (Bunkhumpornpat & Subpaiboonkit 2013, pp. 570 – 575; Gosain & Sardana 2017, pp. 79 – 85; Bunkhumpornpat, et al. 2011, pp. 1 – 4; Meidianingsil, et al. 2017, pp. 1167 – 1171).

The concept of the stacking ensemble method is the specialization of machine learning that takes different machine learning techniques and allows them to vote for an output. The stacking ensemble method has previously been implemented for voting the output using weighted voting and majority voting (Ali, et al. 2015, pp. 211 – 216; Li & Wang 2017, pp. 73 – 77; Dalvi & Vernekar 2016, pp. 1747 - 1751). However, this research study is against the usage of majority voting for a binary classification problem on the bases that a model with 10% predictive accuracy percentage cannot be treated similarly with a model with 90% predictive accurate predictive accuracy percentage when voting the output. For this reason, weighted voting is used for final voting of transactions in this research study. The usage of weighted voting is widespread (Ali, et al. 2015, pp. 211 – 216; Mu, et al. 2005, pp. 2661 – 2666; Li & Wang 2017, pp. 73 - 77).

The weights for classifier models per transaction must be high for a class that performs well, and low for a class that does not perform well. This means that selection of the appropriate weights of votes for each class per classifier is very important. To select the appropriate weights of classifier models, Differential Evolution (DE) method is used

to search for optimum weights is used in this research study. DE is a stochastic method for optimization that is simple in structure but efficient for global numerical optimization (Funaki & Takagi 2011, pp. 287 - 290). Literature evidence on the successful application of DE is plentiful (Bouteldja & Batouche 2017, pp. 1 – 8; Domingo, et al. 2013, pp. 105 – 111; Hui & Suganthan 2013, pp. 135 – 142; Funaki & Takagi 2011, pp. 287 - 290; Goudos, et al. 2016, pp. 1 - 4).

Once the prediction of transaction is made, the model must be evaluated. Various studies have used accuracy score for computing the accuracy of the classifiers (Zaza & Al-Emran 2015, pp. 275 – 279; Nizar, et al. 2008, pp. 1 – 8; Mei & Jiang 2016, pp. 301 – 305; Singh, et al. 2015, pp. 388 - 393). The model accuracy score is the count of correctly predicted transactions over the total count of transactions (Singh, et al. 2015, pp. 388 - 393). In simple terms, the number of fraudulent transactions classified correctly and the number of non-fraudulent transactions classified correctly cannot be derived from the accuracy score.

The actual accuracy score of the model is obtained through the correct prediction of the balance between the percentage of fraudulent and non-fraudulent transactions. The aim is to build a classification model that equally represents both fraudulent and non-fraudulent classes. Accordingly, to evaluate the classification model, this research study computes the confusion matrix evaluation method on the values of fraudulent and non-fraudulent classes. The confusion matrix shows the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) predicted transactions (Rajak & Mathai 2015, pp. 1 - 4).

The number of true positive (TP) transactions is the number of output transactions that the base classifier has predicted as being fraudulent when they were in fact fraudulent. Similarly, the count of true negative (TN) transactions is the count of output transactions that the base classifier has predicted as being non-fraudulent when they were in fact non-fraudulent, the count of false positive (FP) transactions is the count of output transactions that the base classifier has predicted as being fraudulent when they are non-fraudulent, and the count of false negative (FN) transactions is the count of output transactions that the base classifier has predicted as being non-fraudulent when they are in fact fraudulent.The confusion matrix evaluation method has been

applied successfully by (Rajak & Mathai 2015, pp. 1 – 4; West & Bhattacharya 2016, pp. 1796 - 1801).

2.12. Summary

The rate at which fraudsters continue to commit fraudulent transactions given advanced data analytics models to detect fraudulent transactions is growing continuously. This conveys the existence of an unknown gap between the knowledge of customers and the techniques of fraudsters, which fraudsters diligently exploit in order continue to committing fraudulent transactions.

The standard machine learning algorithms that achieve higher accuracies tend to classify all transactions as the majority class that is always non-fraudulent transactions especially when data is huge and fraudulent transactions are low in terms of numbers. This implies poor predictive accuracy of fraudulent transactions (which is the subject of this research study) since the two classes of transactions are overlapping. Consequently, the need for elimination of high variance and biasness of data distribution, tuning of the data models, and combining predictive capabilities of data analytics methods to improve predictive accuracy was identified as a research gap that can be addressed by this research study. The research design and methodology to address the research gap is explained in Chapter 3 of this research study.

3. Chapter 3: Research Methodology

The methodology section describes the procedures that were followed to fulfil the research objectives of this study. After defining the data source systems and collecting data from the source systems, pre-processing of the data, which involved data preparation, missing data values, normalization, feature selection, and imbalance of data distribution sections were undertaken. Lastly, explanatory data analysis for data understanding and for data mining to uncover the unknowns was undertaken prior to data modelling involving base models, weighted voting stacking ensemble method, and differential evolution optimization methods.

The key word algorithm and method are used interchangeably throughout the research study. In the next section, the data source system of the study is discussed.

3.1. Data Source

This research is designed as one possible solutions to big data challenges faced by South African banks. Several e-mails sent to South African banks requesting the dataset. (Pushpa & Malini 2017) have highlighted that banks do not encourage the sharing of the data for experimental purposes due to the confidentiality of credit cards information.

To model, evaluate and deploy the approach proposed in this research study, the Kaggle machine learning competition data repository (Kaggle 2018) was used as a source system through which dataset to achieve research objectives were collected. The dataset was chosen because it is the credit card dataset and the dataset contains the actual transactions performed by credit card users. The link to the Kaggle machine learning competition data repository dataset is as follows:

https://www.kaggle.com/mlg-ulb/creditcardfraud

The Kaggle machine learning competition data repository dataset available for experimental purpose was transformed to hide credit card users' private information. However, the time and the amount were not transformed. The time and the amount features correlate with the South African banking sector's dataset such that the slip of each and every transaction shows the time and the transaction amount. However, the time and the transaction amount may not be sufficient enough to correlate the dataset

from the Kaggle machine learning competition data repository and the dataset generated by the South African banks.In the next section, data preparation for this research study is discussed.

3.2. Data Preparation

The first step involved importing the dataset in csv file format. To import the csv file, a pandas package was used with pd as its alias. When a pandas package was imported, the defined function shown below was used for importing the dataset into Jupyter notebook editor, and the class feature values of the dataset were thereafter separated from the rest of the features.

```
def Import ():
    read = pd.read_csv('creditcard.csv', sep = ',')
    Class = read['Class']
    Data = read.loc[:, read.columns != 'Class']
    return Data, Class
```

The Data and Class of Import function was used to separate the dataset into a testing data and a training data. The function shown below demonstrates how the Data and Class data was used to split the dataset, normalized the Time and the Amount in the dataset, and combined the features of the training dataset with the training dataset class values and called it a Training_data.

```
def split_data(Data, Class):
    X_train, X_test, y_train, y_test = train_test_split(Data, Class,
    test_size=0.3)
    X_train['Time']                                                    =
    StandardScaler().fit_transform(pd.DataFrame(X_train[['Time']]))
    X_train['Amount']                                                  =
    StandardScaler().fit_transform(pd.DataFrame(X_train[['Amount']]))
    Training_data = pd.DataFrame(X_train)
    Training_data['Class'] = y_train
    return X_train, X_test, y_train, y_test, Training_data
```

Since the class distribution was highly skewed, the training dataset was balanced such that there was an equal representation of both classes on the dataset. The Training_data from the above function was used for balancing of the dataset. To balance the dataset, the data observations were separated according to their class values. The function _separator shown below was used to separate the data observations. Whereas the Training_data0 represented the non-fraudulent transactions, which were the majority class data observations, the Training_data1 represented fraudulent transactions which were the minority class data observations.

```
def _separator(Training_data):
    Training_data0 = []
    Training_data1 = []
    for ind in range(len(Training_data)):
        if Training_data.iloc[ind][-1] == 0.0:
            Training_data0.append(Training_data.iloc[ind])
        else:
            Training_data1.append(Training_data.iloc[ind])
    return Training_data0, Training_data1
```

The credit card data contains transactions that were made in September 2013 by the European cardholders. This dataset is made up of 492 fraudulent transactions out of a total of 284 807 transactions that were made in two days. This means that this dataset is highly unbalanced with 0.17% of fraudulent transactions and 99.83% of non-fraudulent transactions. The Training_data0, which constitutes the majority class, was used for down-sampling the data observations while the Training_data1 (i.e. the minority class) was used for oversampling the data observations.

Due to confidentiality of the customers' private information, this dataset was transformed using Principal Component Analysis (PCA) method to contain only numerical input values in order to hide the background information and the features of the dataset. This dataset consists of 28 principal components (V_1, V_2, …, V_28) which are the results of PCA transformation, and the feature Time and Amount that were not transformed.

The dataset consisted of an additional feature called Class, which is the response variable of each transaction made by the customers. The response variable 1 denotes an illegal transaction; on the other hand, the variable response 0 denotes a legal transaction. The two features that were initially not transformed were transformed using the PCA method to normalise all variables of the dataset.

3.3. Missing Values

For the purposes of data quality checking, the dataset was tested for null values and the missing values in this study. The test results from the dataset showed that there were null values or missing values in the data; this is regarded as important for the data analysis. The dataset with no missing values says a lot about the legitimacy of the source systems. Since the dataset is cleaned and pre-transformed, it was not possible to make any conclusions regarding the legitimacy of the source systems.

3.4. Data Normalization

Since some variables have small variance than others, the normalization of data is required to scale variables to have values of between 0 and 1. That being said, the dataset to achieve the research objectives consists of 28 principal component features that were transformed using PCA method, and two features that were not transformed. Thus, the two features which were not transformed were transformed with the PCA method to normalise the entire variables of the dataset.

3.5. Feature Selection

Since the data set is at this stage transformed with PCA and the original feature names of 28 vectors are unknown, the study performed a manual stacking ensemble method on 3 feature selection methods to select the relevant features that improves the results of the data model. The 3 methods of feature selection are univariate selection, recursive feature elimination, and feature importance. The similar features that were obtained by Univariate selection, feature importance, and recursive feature elimination were tested by a logistic regression classification model and evaluated by confusion matrix method.

3.6. Imbalanced data

The dataset was transformed by normalizing all feature vectors such that they have a scale of between 0 and 1. The normalization was undertaken to deal with the issue of feature vectors having different feature scales for solving a single problem. Feature vectors having different feature scales causes the data model to perform poorly.

The data samples from the majority class that have similar variance introduce duplication of data sample. Thus, too many duplicates were eliminated from the dataset. Having too many duplicates of the data sample introduces data model learning one data sample more than the other data samples. The issue of data model learning one data sample many times other data samples introduce skewed learning of data samples. This means that the data model had more knowledge of highly duplicated data sample since their learning was repeated many times. Thus, the duplicate transactions were eliminated to have equal representation of each legal transaction.

To address the issue of skewed distribution: the under-sampling of majority class to eliminate skewed learning (which is formally known as biasness) and preserve as much knowledge as possible was computed together with the Safe-Level SMOTE oversampling method. The Safe-Level SMOTE is an important oversampling of minority class because it does not duplicate minority training samples; it instead introduced new possibilities of the minority class. That is to say, Safe-Level SMOTE introduced new possibilities of the minority class. SMOTE introduced new possibilities of abnormal behaviours of credit card fraudsters.

The dataset with low biasness and low variance eliminates issues such as overfitting the data model, under-fitting the data model, and overlapping data samples. Safe-Level SMOTE oversampling method was implemented as follows:

- The standard deviation median of the continuous features of the class of minority data observations, which in this case were fraudulent transactions, were computed. For nominal features, the median was computed by finding the Euclidean distance between a data observation and its potential nearest neighbours.
- The nearest neighbour between the feature vector and other feature vectors in the class of minority data observations in the continuous feature space was computed. That is to say, for every vector, a median of the standard deviation between a feature vector and its potential nearest neighbour was included.

3.7. Visualisation

PCA is a feature reduction technique that compresses dataset using a linear algebra method. PCA reduction technique is transformed by choosing the number of dimensional vector spaces or principal components in the transformed results. In this study, 2 was selected as the number of components in order to visualize the logistic regression classifier into 2-dimensional vector space. The logistic regression was visualised on balanced class distribution based on features selected from a manual stacked ensemble method.

3.8. Explanatory Data Analysis

This is an important section of the study, which describes an approach to data analysis. The Amount and Time were the only known features of the dataset. V1, V2, …, V28 were the additional features of the dataset that were transformed using PCA method to hide the origin of the dataset. The scatter plot visual illustrated in Figure 3.1 is the correlation between the transaction amount and Time spent on transaction.

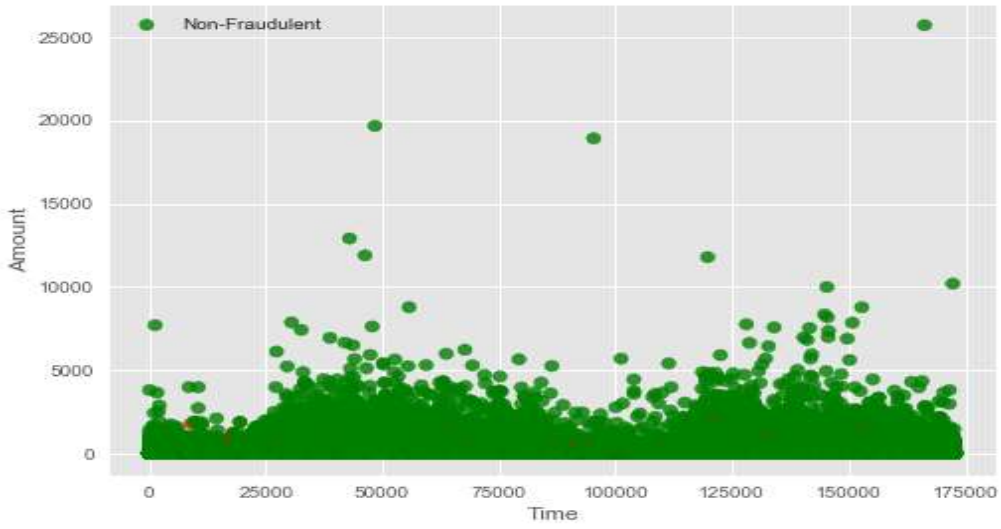


Figure 3.1: Correlation between the transaction amount and Time

The Time vs Amount scatter plot visual reflected the sophistication of credit card fraudsters. The green data points are non-fraudulent transactions, and the red data points are fraudulent transactions. The time spent performing a fraudulent transaction was in the same domain as the time spent performing a non-fraudulent transaction, and the amount spent on fraudulent transaction was in the same domain as the amount spent on non-fraudulent transaction. As observed in Figure 3.1, many of the data points are at y = 0 in relation to time; this suggests that the time feature was not in the transactions performed.

The heat-map of feature correlation (see Figure 3.2) indicates strongly correlated features. The selected features are strongly related and have a correlation value that is greater than 0.4 and -0.4. Using the heat-map, the features found to be strongly correlated were identified as Time & V3, which possess a correlation value of -0.53.

The pictorial presentation of the correlated features provided insight on how fraudulent transaction could be separated from non-fraudulent transaction.

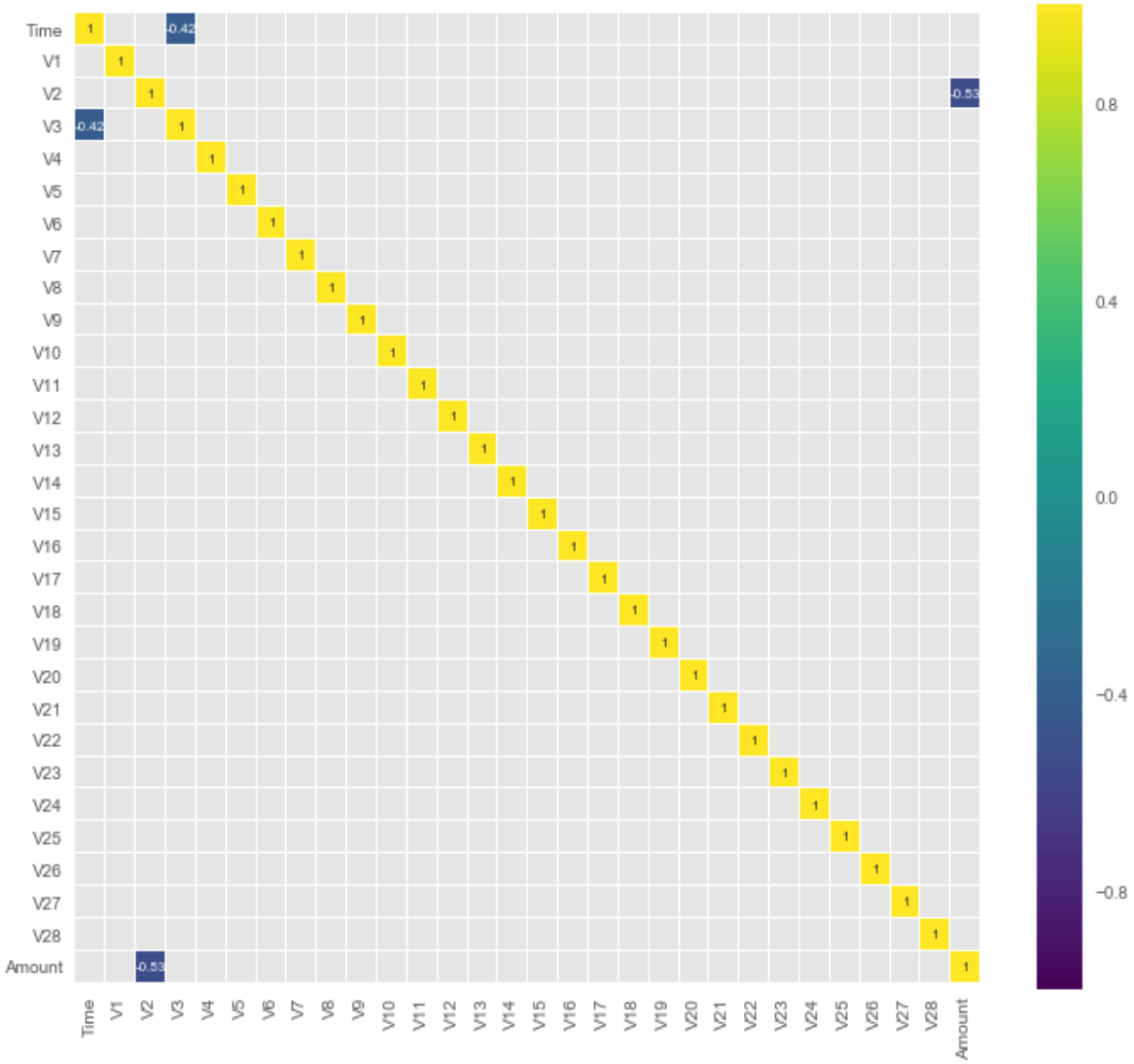


Figure 3.2: Heatmap of Feature Correlation.

The correlation between the amount and the transformed feature V2, which is illustrated in Figure 3.3, shows how some of fraudulent transactions can be separated from non-fraudulent transactions. Whereas the green data points are non-fraudulent transactions, and the red data points are regarded as fraudulent transactions. The positive side of V2 showed more fraudulent transaction, and the negative of V2 showed more non-fraudulent transactions. However, many of the fraudulent data points were at y = 0, suggesting that the fraudulent transactions have less to do with the transaction amount.

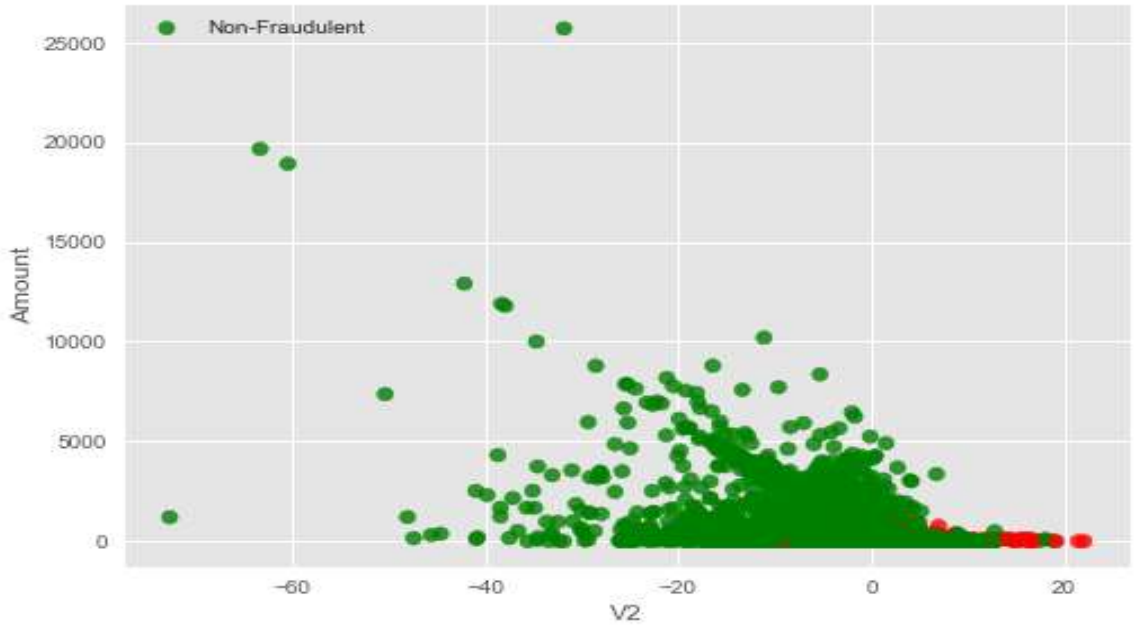


Figure 3.3: Correlation between the transaction amount and V2.

The correlation between the Time and the transformed feature V3, which displayed in Figure 3.4 also paints a picture of how some of fraudulent transaction can be separated from non-fraudulent transactions. As in the Figure 3.3, the green and red data points are non-fraudulent and fraudulent transaction respectively. In addition, while the positive side of V3 showed more non-fraudulent transactions, the negative side of V3 showed a near balance between fraudulent transactions and non-fraudulent transactions. However, a V3 of -5 and -32 and a Time of between 5000s and 3000s and 90000 and 100000 indicated a high number of outliers relating to fraudulent transactions than those of non-fraudulent transactions.

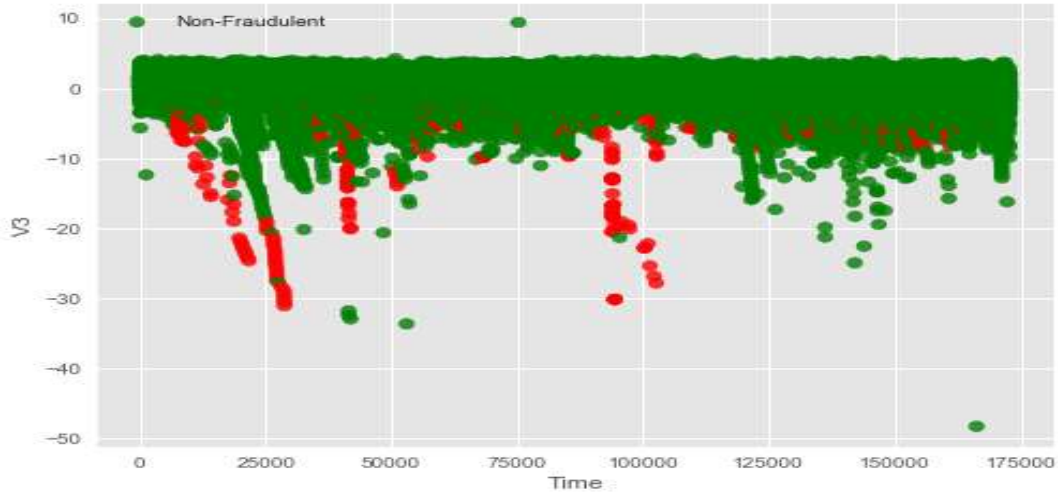


Figure 3.4: Correlation between the V3 and Time.

According to Figure 3.5, which shows a series of correlations between an Amount and all other features of the dataset, indicate a generally linear relationship. Interestingly, a lot of data points were observed at y = 0 and x = 0 indicating the absence of the features. A decision was taken to eliminate the data points that do not add any value to the correlation visuals. However, for the sake of preserving as much information as possible about the transactions, a decision was taken not to eliminate data points at x = 0 and y = 0, since the dataset contains many unknown features.

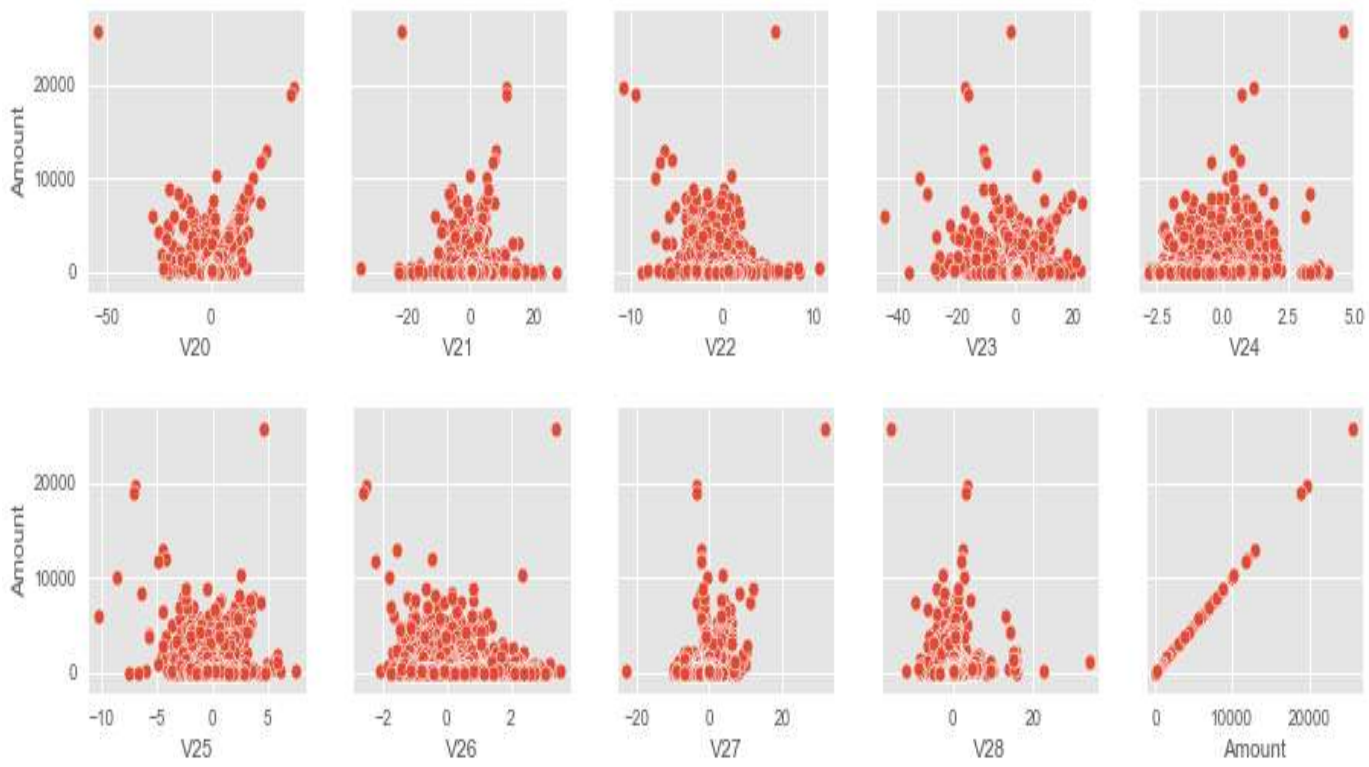


Figure 3.5: Scatter plot correlations between Amount and all other features.

3.9. Data Model Selection and Training

The individual base model classifiers for the stacking model should be powerful and as diverse as possible in order to cover the scope of the problem and as many anomalies as possible within the problem domain. The ANN, SVM, KNN, and the decision tree were chosen to represent the diversity of base models in this study in order to covers the problem domain. The algorithms were chosen because they are popular, powerful, widely used, and vary in performance. The first algorithm that is up for discussion is artificial neural network.

3.9.1. Artificial Neural Network (ANN) Method

The training process adopted for this study used a training sample of size N [$(x_1, y_1,)$, $(x_2, y_2), \ldots, (x_N, y_N)$], where $x_{(1,..,N)}$ denotes credit card transaction performed by customers, and $y_{(1,..,N)}$ denotes the class labels for each transaction performed by the customers (the class labels are either fraudulent or non-fraudulent).

The main technical challenge associated with ANN was to find the number of hidden layers and the training parameters that produces a good performing model. The

number of ANN's hidden layers and training parameters for the best average accuracy was accomplished by running a nested loop that loops through the number of hidden layers and the number of epochs; the best performing number of epochs and hidden layers were chosen to train the ANN model using the entire training dataset.

After training the ANN model with the training dataset, the testing data was used to test how well the classifier could separate fraudulent transactions from non-fraudulent transactions.

3.9.2. Support Vector Machine (SVM) Method

The training process of the research study used the training sample of size N [(x_1, y_1,), (x_2, y_2), …, (x_N, y_N)], where $x_{(1,..,N)}$ denotes credit card transaction performed by customers, and $y_{(1,..,N)}$ denotes the class labels for each transaction performed by the customers (the class labels are either fraudulent or non-fraudulent). The main technical challenge encountered with the support vector machine was to find the best training parameters that produce very good performing support vector machine model.

The grid search method to search for optimal parameters was adopted for this research study. The optimal parameters are C and gamma for best training the model looking at the nature of the credit card dataset. Gamma is a free parameter of RBF (Radial Basis Function) kernel function, and C is a parameter regularization for a soft margin of a cost function. The optimal SVM parameters were used to train the SVM model with the entire training dataset.

After training the SVM model, the testing dataset were used to establish how well the model can separate fraudulent transactions from non-fraudulent transactions.

3.9.3. Decision Tree (DT) Method

The training process of the research study used the training sample of size N [(x_1, y_1,), (x_2, y_2), …, (x_N, y_N)], where $x_{(1,..,N)}$ denotes credit card transaction performed by customers, and $y_{(1,..,N)}$ denotes the class labels for each transaction performed by the customers (the class labels are either fraudulent or non-fraudulent).

The main technical challenge encountered with training the decision tree model was to find the training parameters that produce a good performing model. For decision tree parameter settings, random search and grid search were used, and the search

method that produces good results in a lesser period of time was used to train the entire training dataset.

After the decision tree model was trained with the training data, the testing data was used to establish how well the model could separate fraudulent transactions from non-fraudulent transactions.

3.9.4. k-Nearest Neighbour (KNN) Method

The training process of the research study used the training sample of size N [(x_1, y_1,), (x_2, y_2), …, (x_N, y_N)], where $x_{(1,..,N)}$ denotes credit card transaction performed by customers, and $y_{(1,..,N)}$ denotes the class labels for each transaction performed by the customers (the class labels are either fraudulent or non-fraudulent). The main technical challenge with KNN was to find the value of K that produces a good performing KNN model.

A manual selection of K value from 1 up until the value of 7 was performed, and the best performing K value was selected to train the KNN model using the training data. The Testing data was used to test how well the KNN method can separate fraudulent transactions from non-fraudulent transactions.

3.9.5. Naïve Bayesian (NB) Method

The training process of the research study used the training sample of size N [(x_1, y_1,), (x_2, y_2), …, (x_N, y_N)], where $x_{(1,..,N)}$ denotes credit card transaction performed by customers, and $y_{(1,..,N)}$ denotes the class labels for each transaction performed by the customers (the class labels are either fraudulent or non-fraudulent). The main technical challenge with the Naïve Bayesian method was to find the statistical data distribution method for estimating the probability distribution of the input values when given the training dataset output values.

The normal (gaussian) data distribution method was used for estimating the probability distribution of the input value from the training data output values. The Testing data was used to test how well the NB model could can separate fraudulent transactions from non-fraudulent transactions.

3.9.6. Stacking Model

Stacking ensemble method is a technique that takes the output values of machine learning algorithms defined in subsections 3.9.1 - 3.9.5 to vote the final output of each credit card transaction. In this research study, the voting method used in the stacking ensemble method is the weighted voting.

Weighted voting is a voting system in which different base models are given different weighting for decision making purposes (Smolyakov, 2017). Basically, some base models have more influence than others. For example, one base model may have 57% score for predicting the outcome of a particular dataset, and another base model has achieved a score of 90% for predicting the outcome of the same dataset.

When using a majority voting system, it is realistic to say this will cause conflict in many cases whereby the first base model incorrectly classified a data sample and the second base model correctly classified a data sample. Thus, the majority voting system is not significant when using two base models. In a case where there is a conflict, a third base model is added and it becomes the decider. The higher the predicting score of the decider model the better the chance of the classifier adding value to the decision making of the stacking model.

Majority voting is a voting in which all the base model classifiers have equal voting regardless of their weighting (Smolyakov, 2017). According to the majority voting, a base model with a predictive score of 10% should have equal voting rights as the base model with a predictive score of 98%, which logically does not make sense because the difference between 98% and 10% is 88%, and not 50% used for assigning the weight of the voting.

Unlike the majority voting, every model of weighted voting plays a role in making the final decision on the classification of a data sample. The weighting voting applied in this research study was based on predictive score of each base model. Although the predictive score was used to assign weights of votes per classifier for the binary classes, appropriate selection of the weights of votes for the binary classes was considered important. Thus, appropriate weighting of votes was deemed an optimization issue. The choice of the optimizer was regarded as being important because the chosen optimizer was expected to be able to correct the incorrect weights

of votes for the binary classes. In this research study, differential evolution method was chosen as the optimizer for correcting the incorrect weights of votes.

Differential evolution is a stochastic method of optimization that is simple in structure but efficient for global numerical optimization. Differential evolution optimizes a problem of incorrectly weighting of votes by maintaining a population of candidates with optimal weighting of votes and creates a new candidate with the best score on the optimization of incorrectly weighting of votes (Brest, et al. 2006, pp. 646 - 657; Srinivas, et al. 2018, pp. 216 - 217; Madathil, et al. 2017, pp. 1 - 5). It is expected that the stacking classifier model of the study will make predictions based on weighted voting on unknown credit card transactions.

3.10. Evaluation of Data Models

A good binary classification data model represents both the fraudulent class and the non-fraudulent class equally or nearly equal without any bias. Thus, the true accuracy of the classification data model will be achieved by computing the confusion matrix of fraudulent and non-fraudulent classes. When the true positive value of the confusion matrix of the fraudulent and non-fraudulent transaction classes are equal or close to each other, the accuracy percentage achieved by the model will be regarded as the true accuracy of the model.

The machine learning algorithms discussed in subsections 3.9.1 - 3.9.5 are base models for stacking ensemble method. Each base model was coded from scratch and was computed with optimum hyper-parameter values using the training dataset. The testing dataset was used to assess the performance of each base model.

To assess the performance of the base models, the accuracy function was designed by counting the correctly classified credit card transactions against the total number of transactions. However, the accuracy of the base models does not say much about the equal representation of correctly classified class values. The equal representation of correctly classified class values was achieved by computing the confusion matrix function.

The confusion matrix output had four values, namely the true positive (TP) transactions, the false positive (FP) transactions, the false negative (FN) transactions,

and true negative (TN) transactions. The true positive (TP) transactions arose from the count of output transactions that was predicted by the base classifier as being fraudulent when they were in fact fraudulent. The false positive (FP) transactions are the number of output transactions that was predicted by the base classifier as being fraudulent when they were in fact non-fraudulent. The false negative (FN) transactions are the number of output transactions that was predicted by the base classifier as being non-fraudulent when they are in fact fraudulent. Lastly, the false positive (FP) transactions refers to the number of output transactions that was predicted by the base classifier as being fraudulent when they were in fact non-fraudulent.

The evaluation of base models was based on the base models' higher accuracy percentage while the true negatives (TP) and true positives (TN) were well represented in each base model. The higher percentage of TP and TN that were closer to each other when the false negative (FN) and false positive (FP) percentages were low symbolizes a well performing base model that was not biased. The well performing base models were used to vote the final prediction of each credit card transaction.

The weighted voting method of the stacking ensemble method was used to vote the final output class values. The appropriate weights of the base models were achieved by computing a differential evolution stochastic optimizer method which converges to the optimum solution while searching the search space. The optimum solution were the appropriate weights of the base models. The appropriate weights of the base models were multiplied with the probabilities of each transaction of the base models and a logistic function was used to class the transactions.

A confusion matrix was computed for the weighted voting stacking ensemble method with differential evolution stochastic optimizer method with a view to evaluate how well the class values were represented. The higher percentage of TP and TN that were closer to each other when the false negative (FN) and false positive (FP) percentages were low, thus symbolizing a well performing weighted voting stacking ensemble method with differential evolution stochastic optimizer method that was not biased.

The base models of the stacking ensemble method outlined in subsections 3.9.1 - 3.9.5 was designed from scratch using the object-oriented programming (OOP) data structure. In this research study, the weighted voting stacking ensemble method and

differential evolution stochastic optimization method were also designed from scratch. Thus, the design and documentation of all algorithms was required. The design of all algorithms is well documented in chapter 4.

3.11. Tool Selection

An open source tool called Python was used in this research study for performing data analysis and predicting future outcomes. Tools for data analysis include:

- The computer system/ laptop with enough storage system for installing the requisite programming software and packages for achieving the objectives of this research study. The computer system/laptop was also used for storing and processing the dataset.
- Internet connectivity has proven to be important for downloading and installing all the necessary requisite software and packages for accomplishing the study objectives of this research study. Internet connectivity also proved to be important when downloading the dataset required for computing the research study objectives.
- Internet browser was used for accessing the repository in which the dataset to accomplish the objectives of this study was located. The internet browser was also used for running the Jupyter notebook (i.e. the editor for the python code).
- The Anaconda setup was installed on the laptop. The Anaconda contains the Jupyter notebook, which is the editor through which the python 3 programming language was coded.
- Other relevant packages such as NumPy, pandas, matplotlib, seaborn were installed on the laptop and were used for importing the dataset and for performing mathematical operations and data modelling, data manipulation and data visualization.

3.12. The Final Model

The Figure 3.6 below shows a flow of how the research objectives of this research study are carried out. The shaded dot indicates the begin of the flow chart, the arrows indicate the direction, the circle and square shapes indicate the activities, and the shaded dot with circle around indicates the end of the flow chart.

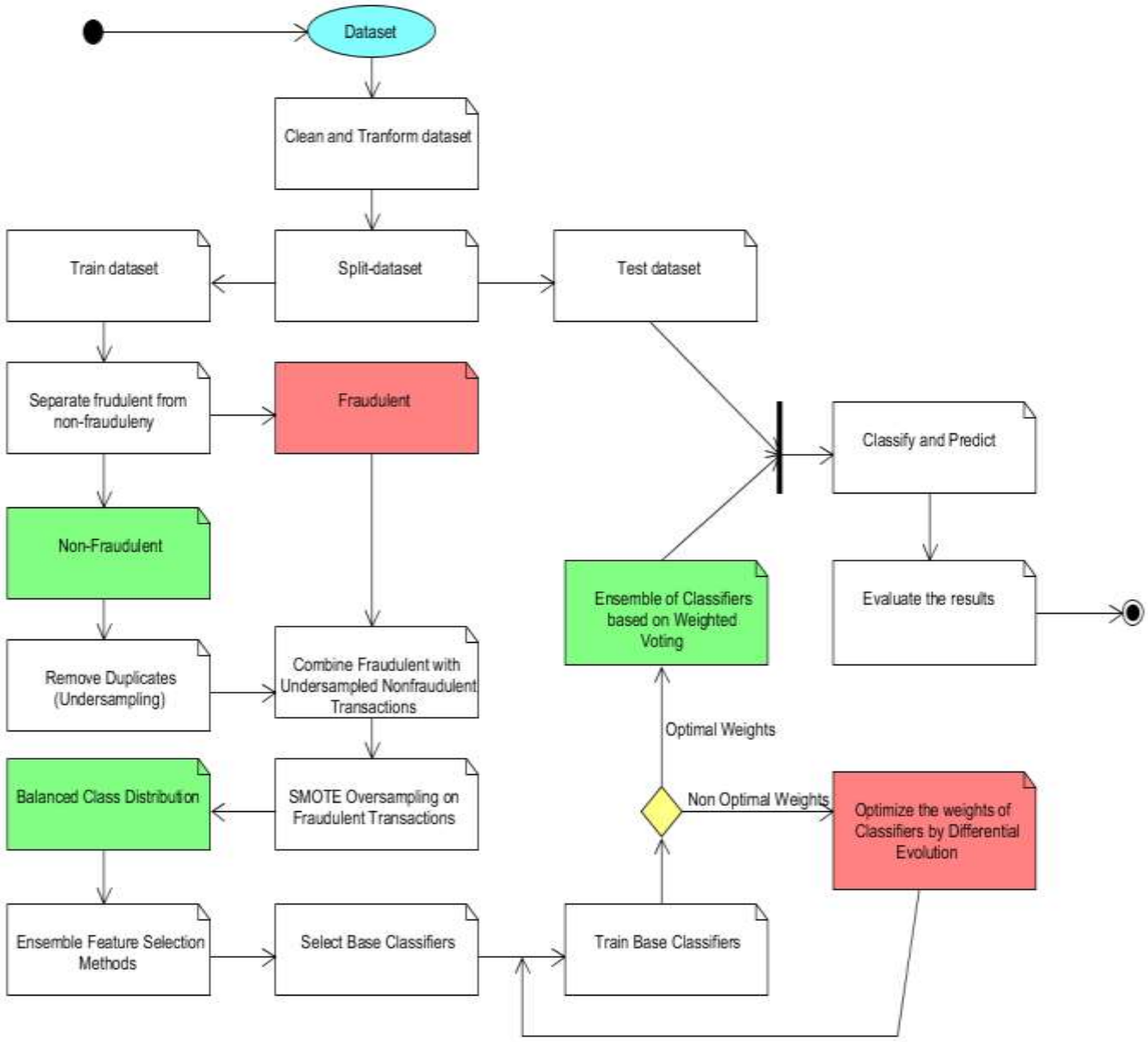


Figure 3.6: Credit card fraud model

3.13. Summary

In this chapter, a detailed description of the research design and methodology have been given. The experimental dataset from Kaggle machine learning competition data repository was used to fulfil the research objectives of this research. This chapter was ended off with the flow chart of how the research objectives of this research are carried out. The experimental algorithms to fulfil the research objectives are discussed in chapter 4 of this research study.

4. Chapter 4: Algorithm Design

An algorithm represents a set of instructions that tell a computer system what to do. In this study, a set of instructions to fulfil the research objectives of this study were defined by the Jupyter notebook editor. The Jupyter notebook file and the dataset were stored in the same folder in order to avoid specifying the path to where the dataset was stored.

Save for the decision tree algorithm, algorithms such as ANN, SVM, KNN, DT and NB were defined from scratch. The algorithms were used as base models of the weighted voting stacking ensemble method. The weighted voting stacking ensemble method was also defined from scratch to vote the final output of each transaction. The weighted voting of the final output for each transaction was computed using the differential evolution stochastic technique. The differential evolution stochastic technique was also defined from scratch with a view to obtain the optimum weights of the base models.

The algorithms were defined to analyse the credit card dataset with the aim of overcoming the challenges of the classifier models. Challenges associated with machine learning classifiers include an inability to classify the fraudulent transaction correctly. Whereas some machine learning classifiers are good for classifying specific portions of the dataset, other machine learning algorithms are good for classifying other portions of the credit card dataset. Thus, weighted voting stacking ensemble method with differential evolution stochastic optimization method was designed to mainly overcome the challenges of the machine learning algorithms for detecting illegal credit card transactions.

In the first section of this chapter, the data balancing methods were discussed. Balancing the dataset for binary classification problem means that two classes of the dataset must have equal representation so as to avoid biasness in the dataset. The data balancing method for down-sampling the majority dataset was for removing the duplicate data observation and randomly sample the data observations. The data balancing method for oversampling the minority class is selected from among SMOTE, SL-SMOTE, Modified SMOTE, and Modified SL-SMOTE. The Modified SMOTE and Modified SL-SMOTE are the algorithms designed in this study for modifying the

performance of the existing SMOTE and SL-SMOTE. The oversampling method with a superior performance was chosen to oversample the minority class of the credit card dataset when balancing the class distribution.

The balancing of the dataset was followed by a feature engineering process. The feature engineering process is a process through which important features for decision making are selected for data modelling. Feature selection was followed by the construction of the weighted voting stacking ensemble model for weight voting the final output transactions. The weighted voting stacking ensemble model was followed by differential evolution method for the optimization of the weights of the stacking ensemble method. In the next sub-section, the balancing of class distribution of the dataset is discussed.

4.1. Balancing the Dataset

This section is focussed on how the class distribution was balanced. To balance the class distribution, the down-sampling of majority class was performed in order to decrease the number of data observation from the class of the majority data observations. The oversampling of data observation from the class of the minority data observations was performed to increase the data observations.

To down-sample the class of the majority data observations, duplicate data observations were removed, and random sampling of the data observations was performed. The algorithm 1 in section 8.4 was used to remove duplicate data observations, and store data observations without duplicates in Training_data0_df. The pseudo-code below is the highlight of the algorithm 1 in section 8.4.

Algorithm 1: Remove Duplicates in the Training Dataset

Input: Training Dataset
Output: Training Dataset without duplicates
Begin

1. **Select** the training dataset
2. Get the size of duplicates
3. Remove the duplicates
4. **Return** the size of the duplicates and the training data without duplicates

End of the function

The Training_data0_df was used for random sampling of data observations. The algorithm 2 in section 8.4 was used to demonstrate how the random data sampling is carried out. The 700 refers to the number of minority class data observations multiplied by two, since the minority class is also be oversampled. The pseudo-code below is the highlight of the algorithm 2 in section 8.4.

Algorithm 2: Random Sampling

Input: Training Dataset
Output: Random Sample
Begin
Initial: reduced = [], index = []

```
For i = 1 to 700 do
    ind = random integer value between 0 and the size of the training dataset
    IF ind not in index
        Then append ind into index
            Append training dataset row at ind into reduced
    End IF
    Else
        i = i – 1
    End Else
Return reduced
```

End of the function

The randomly chosen data observations from Training_data_df were combined with the minority class. The algorithm 3 in section 8.4 was used to show how the Training_data_df and training_data (i.e. the minority class) are combined. The pseudo-code below is the highlight of the algorithm 3 in section 8.4.

Algorithm 3: Combine Training Dataset without Duplicates with Random Sample Dataset

Input: Training Dataset without Duplicates, Random Sample Dataset
Output: Combined Dataset
Begin

1. Combined Dataset = Random Sample Dataset
2. Append Training Dataset without Duplicates into Combined Dataset
3. **Return** Combined Dataset

End of the function

The recombination from the algorithm 3 in section 8.4 involves a combination of Training_data (i.e. the down-sampled dataset) and Training_data1 (i.e. the minority class of data observation). To balance the dataset, the minority class were

oversampled using Safe-Level-SMOTE, which introduced new minority class data observations.

The SL-SMOTE method used the Euclidean-distance to compute the five nearest neighbours of the minority class and called it k. One of the five nearest neighbours was chosen at random, and was used to compute its five nearest neighbours and assigned them to n_neighbors. From k, the Safe-Level-SMOTE counted the number of positive instances and called it SLp. From n_neighbor, the Safe-Level-SMOTE also counted the number of positive instances and called it SLn.

If SLn is not equal to 0, the Safe-Level ratio was computed by dividing SLp by SLn; instances where SLn equalled 0 the Safe-Level ratio were defined as infinity. If Safe Level ratio was not equal to infinity and SLp was not equal to 0, Safe-Level-SMOTE generates feature values for new data observation such that if Safe Level ratio was equal to infinity and SLp was not equal to 0 then feature value gap was equal to 0. If Safe Level ratio was equal to 1, the feature value gap was the randomly chosen value that was located between the value of 0 and 1, and if Safe Level ratio was found to be greater than 1 then the feature value gap was the randomly chosen value that was located between 0 and 1 / (Safe Level ratio). If Safe Level ratio was found to be less than 1, the feature value gap was the randomly chosen value that was located between (1 – Safe Level ratio) and 1.

The feature value n_neighbor at index is i plus (gap multiplied by the difference between the k feature value at index i and n_neighbor feature value at index I) for all features. Feature values are appended to a list called feature, and feature represents a newly generated data observation and is appended to New_data list as a new instance of the minority class. The algorithm 4 in section 8.4 was computed to generate synthetic instances of the Safe Level SMOTE method. The pseudo-code below demonstrates the flow of the Safe Level SMOTE algorithm in section 8.4.

Algorithm 4: Safe Level SMOTE

Input: Minority data observations-T, Size of new instance in percentage-N,
Size of the nearest neighbours-K
Output: I=(N/100)*count(T) – is the newly generated minority class instances of
size I – Synthetic-instances.
Begin

```
No-of-minority = count(T)
No-new-Instances = I
No-of-attributes = count(T[0] – 1
Minority-instances = T
Synthetic-instances= []
For each positive-instance in positive-instances:
    Compute 5 neighbours of positive-instance and call it K.
    Compute 5 neighbours of positive-instance and call it N.
    Count number of positive instances in K and call it SLp.
    Count number of positive instances in N and call it SLn.
    If SLp != 0:
        SL-ratio = SLp/SLn
    Else:
        SL-ratio = np.inf
    If SL-ratio == np.inf AND SLp ==0:
        Continue.
    Else:
        Feature = []
        For index in range(len(np.array(T)[0])):
            If SL-ratio == np.inf AND SLp != 0:
                Gap = 0
            Elseif SL-ratio == 1:
                Gap = np.random.uniform(0,1)
            Elseif SL-ratio > 1:
                Gap = np.random.uniform(0,1 / SL-ratio)
            Elseif SL-ratio < 1:
                Gap = np.random.uniform(1 - SL-ratio , 1)
          difference = nnarray[nn][attribute] –  positive instance[attribute].
          Gap = generate random value between 0 and 1.
          Synthetic-instances.append(positive - instance[attribute]+ difference * gap)
  Append minority class value to Synthetic-instance.
```

End of the function

The oversampled minority class data observations were combined with the down-sampled data observations from the majority class to form a balanced class distribution. However, the objective of SMOTE algorithm was violated by how the algorithm was designed. The design of SMOTE and Safe-Level-SMOTE algorithms, and the algorithms are rectified to meet their objectives are discussed in the next section.

4.1.1. SMOTE vs Safe-Level-SMOTE

The objective of SMOTE (Synthetic Minority Over-sampling Technique) is to create a new data observation between the two existing data observations of the minority class.

Instead of creating new data observation anywhere in the class of the minority data observations, a safe-level smote oversampling technique argues that the new data observation must be created between the two minority class data observations at the safe level.

The SMOTE, SL-SMOTE, modified SMOTE, and the modified SL-SMOTE were used to oversample the data observations of the minority class of the dataset in combination with down-sampling method. Through the down-sampling method, the duplicates were removed and a non-duplicate data sample was randomly chosen from the class of the majority data observations in order to control the best-fit line to an optimum place that equally represents illegal transactions and legal transactions. A dataset with an equal representation of legal transactions and illegal transactions makes it easier for a classification model to thoroughly learn the data of the two classes.

Thus, the SMOTE, SL-SMOTE, modified SMOTE, and the modified SL-SMOTE were tested by running ANN, SVM, NB and KNN algorithms. Whereas the SMOTE method is compared with the Modified SMOTE method in this section, the SL-SMOTE method is also compared with the corresponding Modified SL-SMOTE method. Thereafter, the Modified SMOTE method is compared with the Modified SL-SMOTE method.

The objective of SMOTE algorithm is to create a new data observation between two existing data observations of the minority class (Chawla, et al. 2002, pp. 321 - 357). In this research study, it is argued that the logic used for generating new data instances using the SMOTE and Safe-Level SMOTE does not always generate a new data instance between two given data instances.

The synthetic instance is generated by finding the difference between the attribute values of between the data instance of interest and its chosen nearest neighbour. The difference is multiplied by the gap consisting of a random value chosen between 0 and 1. The multiplication of the difference and the gap is then added to the data instance of interest.

Although this method works well, there are limitations associated with its objective. On the basis of the logic behind the mathematics of the discussed attribute loop of Safe-Level SMOTE method, it is suggested in this study that if two data instances have values of opposite signs then the Safe-Level SMOTE algorithm would not generate a

new data instance that is between the two data instances, which violates the objective of SMOTE algorithm. The mathematical difference operation for the attribute loop of Safe-Level SMOTE changes to the addition operation when the values of the two data instances assume opposite signs.

The Modified SMOTE algorithm shown below is a demonstration of the above claim using the attribute loop of SMOTE. The attribute loop of SMOTE is where the new data instance is generated between two data points. The algorithm 5 in section 8.4 was computed to generate synthetic instances of the SMOTE method. The pseudo-code below demonstrates the flow of the SMOTE algorithm in section 8.4.

Algorithm 5: SMOTE

Input: Minority data observations-T, Size of new instance in percentage-N,
Size of the nearest neighbours-K
Output: I=(N/100)*count(T) – is the newly generated minority class instances of
size I – Synthetic-instances.

```
Begin
    No-of-minority = count(T)
    No-new-Instances = I
    No-of-attributes = count(T[0] – 1
    Minority-instances = T
    Synthetic-instances= []

    For positive-instance in positive-instances:
        Compute 5 neighbours of positive-instance and call it nnarray.
        For index in No-new-instances:
            Choose a random number between 0 and number of neighbours and call it nn.
            For attribute in attributes:
                difference = nnarray[nn][attribute] –positive-instance[attribute].
                Gap = generate random value between 0 and 1.
                Synthetic-instances.append(positive-instance[attribute]+ difference * gap)
    Append minority class value to Synthetic-instance
End of the function
```

4.1.2. Modified SMOTE Algorithm

In this study, a random generation of values between the value of data observation 1 and data observation 2 as a way of ensuring that the newly generated values lie within the boundary of data observations 1 and 2 is proposed. The modified function of the attribute loop of smote is represented by the algorithm 6 in section 8.4. The pseudo-code below demonstrates the flow of the Modified SMOTE algorithm in section 8.4.

Algorithm 6: Modified SMOTE

Input: Minority data observations-T, Size of new instance in percentage-N,
Size of the nearest neighbours-K
Output: I=(N/100)*count(T) – is the newly generated minority class instances of
size I – Synthetic-instances.

Begin

```
No-of-minority = count(T)
No-new-Instances = I
No-of-attributes = count(T[0] – 1
Minority-instances = T
Synthetic-instances= []

For positive-instance in positive-instances:
    Compute 5 neighbours of positive-instance and call it nnarray.
    For index in No-new-instances:
        Choose a random number between 0 and number of neighbours and call it nn.
        For attribute in attributes:
            row.append(np.random.uniform(instances[0][attr], instances[1][attr])
Append minority class value to Synthetic-instance
```

End of the function

4.1.3. Modified Safe-Level-SMOTE Algorithm

By generating data instances within the boundaries of data instances 1 and 2, the objective of the original smote method was achieved and thus gave merit to a comparison of the smote and safe-level smote algorithms. The function definition of the Modified safe-level SMOTE algorithm is defined in algorithm 7 of section 8.4. The pseudo-code below demonstrates the flow of the Modified safe-level SMOTE algorithm in section 8.4.

Algorithm 7: Modified Safe Level SMOTE

Input: Minority data observations-T, Size of new instance in percentage-N,
Size of the nearest neighbours-K
Output: I=(N/100)*count(T) – is the newly generated minority class instances of
size I – Synthetic-instances.

Begin

```
No-of-minority = count(T)
No-new-Instances = I
No-of-attributes = count(T[0] – 1
Minority-instances = T
```

```
5.  Synthetic-instances= []
6.  For positive-instance in positive-instances:
7.     Compute 5 neighbours of positive-instance and call it K.
8.     Compute 5 neighbours of positive-instance and call it N.
9.     Count number of positive instances in K and call it SLp.
10.    Count number of positive instances in N and call it SLn.
11.    If SLp != 0:
12.       SL-ratio = SLp/SLn
13.    Else:
14.       SL-ratio = np.inf
15.    If SL-ratio == np.inf AND SLp ==0:
16.       Continue.
17.    Else:
18.       Attribute = []
19.       For index in range(len(np.array(T)[0])):
20.          If SL-ratio == np.inf AND SLp != 0:
21.             Gap = 0
22.          Elseif SL-ratio == 1:
23.             Gap = np.random.uniform(0,1)
24.          Elseif SL-ratio > 1:
25.             Gap = np.random.uniform(0,1 / SL-ratio)
26.          Elseif SL-ratio < 1:
27.             Gap = np.random.uniform(1 - SL-ratio , 1)
28.          Attribute.append(np.random.uniform(nnarray[nn][attribute],positiveinstances[attribute])
29.          Attribute.append[-1] = 1.0
30.          Synthetic-instance.append(Attribute)
31. Append minority class value to Synthetic-instance.
End of the function
```

From the algorithm described above, it can be seen that the new data instances that are generated follow the same logic as the above-mentioned smote method, except the fact that the newly generated data values in this case are generated at the safe level of SMOTE. In addition, to preserve the primary objective of the SMOTE algorithm, the new data values Safe-Level SMOTE must be generated between the data instances at the safe-level of SMOTE. The use of the modified Safe-Level SMOTE algorithm in combination with the down-sampling method that removes duplicates and randomly chooses non-duplicate data sample from the majority class, were used to generate a balanced class distribution. A balanced class distribution was used for feature selection process. In the section that follows, the manner in which the feature selection process is being performed is discussed.

4.2. Feature Selection Method

The balanced class distribution dataset is used for feature selection. The feature selection is performed using the manual stacking ensemble method of Univariate Selection, Recursive Feature Elimination, and Feature Importance. The three feature

selection algorithms were computed, and the **Time** column of the dataset appeared not to add any positive value to the decision making of base models. The **Time** column was eliminated from the data and the performance of the base classifiers of stacking ensemble method was improved significantly. Other features of the dataset were removed individually, and the performance of base models did not improve significantly. The three feature selection methods, namely Univariate Selection, Recursive Feature Elimination and Feature Importance, are discussed in the sub-sections that follows.

4.2.1. Univariate Selection Method

The Univariate Selection Method is performed using the selectKbest method to select the best features based on the f_classification method. The value of k (i.e. the number of features) was set at 30 since there were 30 features of the dataset. The fit.scores was used to get scores for each feature of the dataset. The scores in relation to features were sorted in order to rank the features according their importance. The function definition of the Univariate Selection Method is defined in algorithm 8 of section 8.4. The following pseudo-code shows the flow of how the Univariate Selection Method was defined in section 8.4.

Algorithm 8: Univariate Selection Method

Input: Features, Class
Output: Univariate Scores
Begin

1. Test = pass score function as f_classif and number of input features to SelectKBest
2. Fit = fit the input features and class values
3. Scores = get scores
4. Append sorted scores in descending order into Univeriate_Scores
5. **Return** Univeriate_Scores

End of the function

4.2.2. Recursive Feature Elimination Method

In this research study, the Recursive Feature Elimination Method was used in combination with the logistic regression binary classification method since the balanced class distribution dataset was separable. The function definition of the Recursive Feature Elimination Method is defined in algorithm 9 of section 8.4. The

following pseudo-code shows the flow of how the Recursive Feature Elimination Method was defined in section 8.4.

Algorithm 9: Recursive Feature Elimination Method

Input: Features, Class
Output: Recursive Scores
Begin
1. Model = Logistic Regression
2. RFE = pass the model to Recursive Feature Elimination Method
3. Fit = fit the input features and class values to the RFE
4. Ranking = rank the scores
5. Append sorted scores in descending order into Recursive_Scores
6. **Return** Recursive_Scores

End of the function

4.2.3. Feature Importance Method

The third feature selection method is Feature Importance. In this research study, the Feature Importance was used in combination with the ExtraTreeClassifier method. The ExtraTreeClassifier is used for binary classification, and the features of the dataset were ranked according their importance. The function definition of the feature importance Method is defined in algorithm 10 of section 8.4. The following pseudo-code shows the flow of how the feature importance Method was defined in section 8.4.

Algorithm 10: Feature Importance Method

Input: Features, Class
Output: Recursive Scores
Begin
1. Model = Extract Tree Classifier
2. Fit = fit the input features and class values to the Model
3. Scores = generate feature importance
4. Append sorted scores in descending order into Importance_Scores
5. **Return** Importance_Scores

End of the function

In this research study, the dataset of the selected features was used for the construction of the data models for binary classification. The weighted stacking ensemble method of classifiers is proposed in this research study as the data model

for the detection of malicious activities. The construction of the weighted stacking ensemble method is discussed in the next section.

4.3. Weighted Stacking Ensemble Method of Classifiers

The stacking ensemble method adopted for this study was performed by voting the output of a data instance through the use of various classification methods such as ANN, SVM, KNN, NB and DT for the prediction of the output class of a given data instance. The stacking ensemble method uses the weighted voting method for decision making. The weights of base models were generated by the differential evolution technique. Differential evolution is an optimization technique for finding optimum weights of base models; and it is discussed in detailed later in this chapter. The definition of the various classification methods such as Artificial Neural Network (ANN), Support Vector Machine (SVM), k-Nearest Neighbours (KNN), Naïve Bayesian (NB), and Decision Tree (DT) methods are outlined in detail in the section that follows.

4.3.1. Artificial Neural Network (ANN) Method

The ANN method is illustrated using a data structure called class. The class name for the ANN method is NNClassifier, which contains a constructor to initialize values. The Table 8.1 in section 8 shows the names and descriptions of the class properties of the NNClassifier class. The class definition of the ANN Method is defined in algorithm 11 of section 8.4. The following pseudo-code shows the flow of how the ANN Method was defined in section 8.4.

Algorithm 11: Artificial Neural Network (ANN)

Input: Number of Features, Number of Classes, Number of Hidden Units, L1 regularization, L2 regularization, Number of Epochs, Learning Rate, Number of Batches, Random Seed
Output: Predicted Output
Begin

1. **Function** for **random initialisation of weights** (input layers **(W1)** and inner layer **(W2)**)
2. **Function** for adding the **bias unit** (value = 1)
3. **Function** to define the sigmoid/logistic function: $\frac{1}{1+ e^{-(w^T x)}}$
4. **Forward feed on the network.**
   Input: input features.
   Net_input = **add bias unit** to the first layer of the network
   Net_hidden = dot product of input layer and Net_input ($w1^T x$)
   Act_hidden = apply sigmoid function to Net_hidden
   Act_hidden = add bias unit to Act_hidden

Net_out = dot product of inner layer (w2) and Act_hidden
Act_out = apply sigmoid function to Net_out
Return Net_input, Net_hidden, Act_hidden, Net_out, Act_out

5. **Function** for defining the derivative of Sigmoid function: $\frac{1}{1+e^{-(w^T x)}}(1+\frac{1}{1+e^{-(w^T x)}})$
6. **Backpropagation of the network.**
   Input: Net_input, Net_hidden, Act_hidden, Net_out, Act_out, Class values (y)
   Sigma3 = Act_out – y
   Net_hidden = add bias unit to the Net_hidden
   sigma2 = ($w2^T$sigma3) * sigmoid_prime(net_hidden)
   grad1 = dot product of sigma2 and Net_input
   grad2 = dot product of sigma3 and Act_hidden
   Return grad1, grad2
7. **L2 Regularization function:**
   Input: lambda, w1, w2, length (m)
   Return $\left(\sum_{k=0}^{n}\sqrt{w1}+\sum_{k=0}^{n}\sqrt{w2}\right)/\frac{\lambda}{2\mathrm{m}}$
8. **Function for defining the Loss/Cost function.**
   Input: predicted values (output), actual output values (y)
   For i=0 to length of class values (y)
      if y = 1.0
         sumEntropy += (-log(output[i]))
      else
         sumEntropy += (-log(1 - output[i]))
   Return sumEntropy
9. **Function for error rate**
   Input: predicted values (output), actual output values (y)
   L2_term = L2 Regularization function
   Error = Cross_entropy(output, y) + L2_term
   Return 0.5 * mean(Error)
10. **Backpropagation to update the weights**
    Input: predicted values (output), actual output values (y)
    Net_input, Net_hidden, Act_hidden, Net_out, Act_out = forward feed function(output)
    grad1, grad2 = self._backward(net_input, net_hidden, act_hidden, act_out, y)
    grad1 += w1 + L2
    grad2 += w2 + L2
    Error = Error(y, Act_out)
    Return Error, grad1, grad2
11. **Function** for predictions
    Input: input features.
    Net_input, Net_hidden, Act_hidden, Net_out, Act_out = forward feed function(output)
    Return Maximum Likelihood Estimator on Net_out
12. **Decision Making Function (Softmax)**
    Input: inputs
    Return $\frac{e^{(inputs)}}{\sum_{k=0}^{n} e^{(inputs)}}$
13. **Class probability prediction function**
    Input: input features.
    Net_input, Net_hidden, Act_hidden, Net_out, Act_out = forward feed function(output)
    Return [softmax(i) for i = 0 to length(net_out)]
14. **Function to fit the model**
    Input: input features(x) and output values (y)
    y = apply function to_categorical to y
    for i = 0 to the number of epochs
       error, grad1, grad2 = backpropagation to update the weights(x, y)

```
            w1 -= learning rate * grad1
            w2 -= learning rate * grad2
            print('>epoch=%d, error=%.3f'%(epoch, error))
```

15. **Function to display the score**
    Input: input features(x) and output values (y)
    y_hat = predict (x)
    Return sum(y == y_hat) / length(x)

End of the Class

4.3.2. Support Vector Machine (SVM) Method

In this research study, the SVM method is illustrated using a data structure called class. The class name for the SVM method is Support_Vector_Machine, and contains a constructor to initialize values. The Table 8.2 in section 8 shows the names and descriptions of the class properties of the Support_Vector_Machine class. The class definition of the SVM Method is defined in algorithm 12 of section 8.4. The following pseudo-code shows the flow of how the SVM Method was defined in section 8.4.

Algorithm 12: Support Vector Machine (SVM)

Input: Number of Features, Number of Classes, Number of Epochs, Learning Rate, Random Seed
Output: Predicted Output
Begin

1. **Function** for **random initialisation of weights** (input layers **(W1)** and inner layer **(W2)**)
2. **Function** for adding the **bias unit** (value = 1)
3. **Function** for formatting class values to float number (format_y)
4. **Function for predicting the probability (predict_proba)**
   Input: input features(x) and output values (y)
   Net_input = add bias unit to the input features
   Weighted = dot product of weights (w) with Net_input
   Format_y = format_y(y)
   X_pred = dot product of weights (w) with format_y
   Return X_pred
5. **Function for prediction**
   Input: input features(x) and output values (y)
   Proba = predict_proba(x, y)
   Pred = [maximum likelihood estimator (value) **for** value in Proba]
   Return Pred
6. **Function for calculating the score**
   Input: input features(x) and output values (y)
   Pred = predict(x, y)
   Count = 0
   For i = 0 to length of Pred
   If pred[i] == list(y)[i]
   Count += 1
   Return Count / length of y
7. **Function to calculate highest class probability**
   Input: Predicted list (pred_list) and Predicted probability list (pred_proba_list)

Proba = []
For each Index and value in pred_list
Append pred_proba_list[Index][value] into Proba
Return Proba

8. **Function to train the dataset**
Input:x_train, y_train
For each epoch in epochs
Pred = predict_proba(x_train, y_train)
For ind = 0 to length of Pred:
If Pred[ind][1] >= 0
W = w + learning rate * (-2 * $\frac{1}{\text{epochs}}$ * w)
Else
W = w + learning rate * y_train[ind] *(x_train[ind]) - (-2 * $\frac{1}{\text{epochs}}$ * w)
Return self

End of the Class

4.3.3. k-Nearest Neighbours (KNN) Method

In this study, the KNN method is illustrated using a data structure called class. The class name for the KNN method is k_Nearest_Neighbors, and contains a constructor to initialize values. The Table 8.3 in section 8 shows the names and descriptions of the class properties of the k_Nearest_Neighbors class. The class definition of the KNN Method is defined in algorithm 13 of section 8.4. The following pseudo-code shows the flow of how the KNN Method was defined in section 8.4.

Algorithm 13: K-Nearest Neighbour (KNN)

Input: Row length, neighbours, TrainDataX, TrainDataY, Random Seed
Output: Predicted Output
Begin

1. **Function** for **calculating the distance:** $(row1\_value - row2\_value)^2$
2. **Function** for calculating the Euclidean distance:
Input: row1, row2
Distance = 0
For ind = 0 to length of the row
Distance += distance(row1, row2)
Return the square root of the Distance
3. Function to calculate the **nearest neighbours (getNeighbors)**
Input:TrainDataX
k-nearest_neighbour = []
for ind = 0 to length of TrainDataX
dist = Euclidean_Distance(row1[ind], row2[ind])
Append (ind, dist) into k-nearest_neighbour
Sort the dist in the k-nearest_neighbour
Return top 5 nearest distances
4. **Function** for predicting the transactions
Input:TestRow

```
        k-nearest_neighbours = getNeighbors(TestRow)
        for each neighbour in k-nearest_neighbours:
            count0, count1 = 0, 0
            if TrainDataY[Neighbor[0]] = 0:
                count0 += 1
            elif TrainDataY[Neighbor[0]] = 1:
                count1 += 1
        if count0 > count1:
            predictions = 0
        else
            predictions = 1
        Return predictions
    5. Function to get the predictions
        Input: TestDataX
        Predictions = []
        For each  TestRow in TestDataX
            Append predicted values into Predictions
        Return Predictions
    6. Function to return the Class probability
        Input: TestDataX
        Percentage = []
        For each TestRow in TestDataX
             k-nearest_neighbours = getNeighbors(TestRow)
             count0, count1 = 0, 0
             for each Index and Neighbour in k-nearest_neighbours
                  if  TrainDataY[Index] = 0.0:
                      count0 += 1
                  elif TrainDataY[Index] = 0.0:
              count1 += 1
          if count0 > count1:
                Percentage.append(count0/5)
          Else
            Percentage.append(count1/5)
        Return Percentage
    7. Function to get an Accuracy
        Input: TestDataY, predictions
        Correct = 0
        For y = 0 to length of TestDataY
             If TestDataY[y] = predictions[y]
                Correct += 1
        Return (Correct / length of TestDataY) * 100
End of the Class
```

4.3.4. Naïve Bayesian (NB) Method

In this research study, the NB method is illustrated using a data structure called class. The class name for the NB method is Naïve_Bayesian, and it contains a constructor to initialize values. The Table 8.4 in section 8 shows the names and descriptions of the class properties of the Naïve_Bayesian class. The class definition of the NB

Method is defined in algorithm 14 of section 8.4. The following pseudo-code shows the flow of how the NB Method was defined in section 8.4.

Algorithm 14: Naïve Bayesian (NB)

Input: TrainDataX, TrainDataY, Random Seed
Output: Predicted Output
Begin

1. **Function** to calculate the **mean:** sum(numbers) / length of numbers
2. **Function** to calculate the **standard deviation:** $\sum_{k=0}^{n} \frac{(x-mean)^2}{n}$
3. **Function** to calculate the **Gaussian Distribution (CalculateProbability):**
   Input: RowValue, mean, standard deviation
   Exponent = $\frac{(RowValue-mean)^2}{2(standard\ deviation)^2}$
   Return $\frac{1}{\sqrt{2*pi}}\ e^{Exponent}$
4. **Function** to summarise the input data:
   Input: TrainData
   Summary = mean(TrainData), standard deviation(TrainData)
   Return Summary
5. **Function** to separate Transaction by class values (seperateByClass):
   Input: TrainDataX
   Seperated = {}
   For i = 0 to length of TrainDataX
   Vector = TrainDataX[i]
   If TrainDataX[i] not in Seperated
   Separated[TrainDataY[i]] = []
   Append vector into Seperated
   Return Seperated
6. **Function** to summarise values by class (SummarizeByClass):
   Separated = seperateByClass()
   Summaries = []
   For each classvalue and instances in Seperated
   Summaries[classValue] = summarise(instances)
   Return Summaries
7. **Function** to calculate the class probability:
   Input: TestDataInstance
   Probabilities = []
   Summaries = summarizeByClass()
   For each classValue, classSummaries in Summaries
   Probabilities[classValue] = 1
   For ind = 0 to length of classSummaries:
   Mean, standard deviation = classSummaries[ind]
   RowValue = TestDataInstance[ind]
   Probabilities[classValue] *= CalculateProbability()
   Return Probabilities
8. **Function** to predict the output of the transaction:
   Input: TestDataInstance
   Probabilities = calculateClassProbabilities(TestDataInstance)
   bestLabel, bestProb = None, -1
   for each classValue, probability in probabilities()
   if bestLabel is none or probability > bestProb

```
            bestProb = probability
            bestLabel = classValue
    Return bestLabel
9.  Function to get Predictions
    Input: TestDataX
    Predictions = []
    For i = 0 to length of TestDataX
        Result = predict(TestDataX[i])
        Append Result in predictions
    Return Predictions
10. Function to return probabilities
    Input: TestDataX
    Probas = []
    For each TestDataInstance in TestDataX
        Proba = CalculateClassProbability(TestDataInstance)
        Proba = maximum likelihood estimator (Proba)
        Append Proba in Probas
    Return Probas
11. Function to get an Accuracy
    Input: TestDataY, predictions
    Correct = 0
    For y = 0 to length of TestDataY
        If TestDataY[y] = predictions[y]
           Correct += 1
    Return (Correct / length of TestDataY) * 100
End of the Class
```

4.3.5. Decision Tree (DT) Method

The last and the final base model of stacking ensemble, the DT method, is illustrated using a data structure called class. The class name for the DT method is Decision_tree, and contains a constructor to initialize values. The Table 8.5 in section 8 shows the names and descriptions of the class properties of the Decision_tree class. The class definition of the DT Method is defined in algorithm 15 of section 8.4. The following pseudo-code shows the flow of how the DT Method was defined in section 8.4.

Algorithm 15: Decision Tree (DT)

Input: Gini_Criterion, Max_depth, Min_samples_leaf, Entropy_Criterion, Random Seed
Output: Predicted Output
Begin

1. **Function** to train the dataset using Gini
   Input: criterion, random state, max depth, min sample leaf, TrainX, TrainY
   Clf_gini = decisionTreeClassifier(Input)
   Fit Clf_gini model with TrainX and TrainY
   Return Clf_gini
2. **Function** to train the dataset using Entropy
   Input: criterion, random state, max depth, min sample leaf, TrainX, TrainY
   Clf_entropy = decisionTreeClassifier(Input)
   Fit Clf_entropy model with TrainX and TrainY
   Return Clf_entropy
3. **Function** to predict the probability
   Input: TestX, clf_object
   Predict_prob = clf_object.predict(TestX)
   Return Predict_prob
4. **Function** to make predictions
   Input: TestX, clf_object
   Y_pred = clf_object.predict(TestX)
   Return Y_pred
5. **Function** to calculate the probability
   Input: pred_list, pred_proba_list
   Proba = []
   For each index and value in pred_list
   Append pred_proba_list[index][value] into Proba
   Return Proba
6. **Function** to get an Accuracy
   Input: TestDataY, predictions
   Correct = 0
   For y = 0 to length of TestDataY
   If TestDataY[y] = predictions[y]
   Correct += 1
   Return (Correct / length of TestDataY) * 100

End of the Class

The weighted stacking ensemble method of the above defined base models was performed by voting out the output of any given credit card transaction as to whether the transaction is illegal or not. The output of the base models was used to generate a new dataset. The new dataset was stored in a data dictionary consisting of columns as names of base models, and rows as the predicted output of base models. The data dictionary of the new dataset is discussed in the next section.

4.4. Predictions Data Dictionary

The weighted voting Stacking Ensemble method was achieved by creating the data dictionary of base models that contain predicted values of each credit card transaction,

and the new feature names as the name of base models. The data dictionary is named Model_predictions and is defined as follows:

```
Model_predictions = {'ANN':nn.predict(X_test)
                     'SVM':nnSVM._predict(X_test, y_test)
                     'KNN':nnKNN._getPredictions(np.array(X_test))
                     'DT':nnDT._prediction(X_test, clf_object)
                     'NB':nnNB._getPredictions(X_test)}
```

The nn, nnSVM, nnKNN, nnDT, and nnNB are the class objects of the base models, which are defined in section 4.4. The predict, _predict, _getPredictions, and _prediction are the property of the class objects. The testing dataset was supplied to the prediction class properties of base models in order to get the predicted values.

Once all the base models had generated the predicted values per credit card transaction, the voting method was required to generate the final prediction output. The voting method used in the stacking Ensemble method of the base data models is discussed in the next section.

4.5. Stacking Ensemble of Base Models

The voting class named Voting, which is defined below, allows base models to vote the final output of each transaction of credit card data. The final output value of each transaction can either be classified as 1 to denote a fraudulent transaction or 0 to denote a non-fraudulent transaction. The voting method that has been adopted for this study is based on the principle of majority rule, which means that the class with the majority of votes in respect of base models is the final classification of the transaction.

The Table 8.6 in section 8 shows the names and descriptions of the class properties of the Voting class. The class definition of the Voting Method is defined in algorithm 16 of section 8.4. The following pseudo-code shows the flow of how the Voting Method was defined in section 8.4.

Algorithm 16: Stacking Ensemble Method

```
Input:
Output: Predicted Output
Begin
    1. Function to return voted class (Mode_Fun)
       Input: List
       Predict, Count0, Count1 = 0, 0, 0
       For each value in the List
            If value = 0
                Count0 += 1
           Esif value = 1
                Count += 1
       If Count0 > Count1
           Predict = 0
       Else
           Predict = 1
       Return Predict
    2. Function to return list of predicted values
       Input: List of Base model predicted values (Model_pred_A)
       Predictions = []
       For each Row in Model_pred_A
           Mode =  Mode_fun(Row)
           Append Mode into Predictions
       Return Predictions
    3. Function to get an Accuracy
       Input: TestDataY, predictions
       Correct = 0
       For y = 0 to length of TestDataY
            If TestDataY[y] = predictions[y]
              Correct += 1
       Return (Correct / length of TestDataY) * 100
    4. Function to return unmatched Indexes
       Input: TestDataY, Predictions
       Incorrect = []
       For ind = 0 to length of TestDataY
           If TestDataY[ind] is not equal to Predictions[ind]
               Append ind into Incorrect
       Return Incorrect
End of the Class
```

To enhance the performance of the voting method, the weighted voting technique is required. In this study, the weighted voting method was performed using the predicted class probabilities of output values generated by the base models. Probability is known to be the number between 0 and 1 indicating the likelihood of an event occurring, and class value is the number that represents the class in which the probability value is closer to. This means that, in this study, the predicted class probability of an output class was used for weighted voting method.

For performing weighted voting method, a data dictionary of predicted class probabilities of each base model is required. The data dictionary of predicted probabilities of base models defined in sub-section 4.4, is discussed in this next section.

4.6. Probability Data Dictionary

The use of the Stacking Ensemble method in combination with weighted voting of the base data models involves creating a new data dictionary of base models that is composed of class probability of the predicted value of each credit card transaction. The data dictionary name is Model_Probability and is defined as follows:

```
Ann_pred = nn.predict(X_test)
Ann_pred_Avg = nn.predict_proba(X_test)
ANNProbability = nn.probability(Ann_pred, Ann_pred_Avg)

SVM_pred_Avg = nnSVM._predict_proba(X_test, y_test)
SVM_pred = nnSVM._predict(X_test, y_test)
SVMProbability = nnSVM.probability(SVM_pred, SVM_pred_Avg)

KNNProbability = nnKNN._probability(np.array(X_test))

clf_object = nnDT._train_using_entropy(Xs, Ys)
DT_pred_proba = nnDT._pred_proba(X_test, clf_object)
DT_pred = nnDT._prediction(X_test, clf_object)
DT_pred = [1 if i == 1.0 else 0 for i in DT_pred]
DTProbability = nnDT.probability(DT_pred, DT_pred_proba)

NBProbability = nnNB.probability(X_test)

Model_Probability= {'ANN':ANNProbability,
                    'SVM':SVMProbability,
                    'KNN':KNNProbability,
                    'DT' :DTProbability,
                    'NB' :NBProbability }
```

The nn, nnSVM, nnKNN, nnDT, and nnNB are the class objects of the base models defined in section 4.4. The Model_Probability is the new data dictionary that is composed of predicted class probability and base model names.

The Differential Evolution Optimization technique is used for generating the optimum weights for base model algorithms in decision making, and it is discussed in the next section.

4.7. Differential Evolution Optimization Method

The weight voting of the stacking ensemble method is optimised using the Differential Evolution technique. Differential Evolution is a stochastic technique that is powerful and efficient over a continuous space for solving differentiable and non-linear optimization problems. In this research study, the Differential Evolution method in this study was used for searching for global optimum weights from a defined search space.

The class definition of the Differential Evolution Method is defined in algorithm 17 of section 8.4. The following pseudo-code shows the flow of how the Differential Evolution Method was defined in section 8.4.

Algorithm 17: Differential Evolution Optimization Method

Input: TrainDataX, population_size, Scaling_Factor, Max_Iterator, Crossover_Probability, random_seed
Output: Predicted Output
Begin

1. **Function** to define the bounding of the search space (Bounds)
   Input: TrainDataX
   Bounds = []
   min = minimum values of all input features
   max = maximum values of all input features
   Append min and max into Bounds
   Return Bounds
2. **Function** to ensure the bounds of the search space
   Input: vector, bound
   New_vector = []
   For ind = 0 to length of vector
   If lower bound[ind] >= vector[ind]
   Append lower bound[ind] into New_vector
   If upper bound[ind] <= vector[ind]
   Append upper bound[ind] into New_vector
   If lower bound[ind] <= vector[ind] and upper bound[ind] >= vector[ind]
   Append vector[ind] into New_vector
   Return New_vector
3. **Function** to define the cost function
   Input: input(x)
   Sums = sum([x_i^2 **for** i=0 to length of x])
   Return Sums
4. **Function** to select a sample population size
   Input: population size (pop_size)
   Bounds = Bounds()
   Population = []
   For Index=0 to length of pop_size
   Individual = []
   For each value1 and value2 in Bounds

```
                Append random value between value1 and value2 into Individual
            Append Individual into Population
    Return Population
5. Function to select random values
    Input: lengthList
    Count, RandomList = 0, []
    While count < LengthList
        RandomNumber = Random value between 0 and Population size
        If RandomNumber not in RandomList
            Append RandomNumber into RandomList
            Count += 1
        Else
            Count += 0
    Return RandomList
6. Function to define Mutation
    LengthList = 3
    RandomList = 3 random integer values between 0 and the length of the population
    X_1 = Population value associated with the integer value 1
    X_2 = Population value associated with the integer value 2
    X_3 = Population value associated with the integer value 3
    Diff = X_1 - X_2
    V_donor = X_1 + Scaling_Factor * Diff
    V_donor = Apply Ensure bound function on V_donor
    Return V_donor
7. Function to define Crossover
    V_Targets = Population()
    V_donor = Mutation()
    V_trials = []
    For V_Target in V_Targets:
        V_trial = []
        For Index = 1 to length of the V_Target
            Crossover = random value between 0 and 1
            If crossover < Crossover_Probability
                Append V_donor[Index] into V_trial
            Else
                Append V_Target[Index] into V_trial
        Append V_trial into V_trials
    Return V_trials
8. Function to return Scores
    Gen_Scores = []
    V_trials = Crossover()
    V_Targets = Population()
    For i=0 to Max_Iterator
        Trial_Costs = Cost_Func of V_trials
        Target_Cost = Cost_Func of V_target
        For Index = 1 to length of Trail_Costs
            If Trial_Costs[Index] < Target_Costs[Index]
                Population[Index] = V_trials[Index]
                Append Trial_Costs[Index] into Gen_Scores
            Else
                Append Target_Costs[Index] into Gen_Scores
    Return Gen_Scores
9. Function to define best population
    Gen_Scores = Scores()
```

```
        Gen_min = min(Gen_Scores)
        Gen_Sol = get population value associated with the minimum score(Gen_min)
        Print('Best Generation: ', Gen_min)
        Print('Best Solution: ', Gen_Sol)
        Return Gen_Sol
    10. Function to define Logistic/Sigmoid function: 1 / (1+ e^-(w^T x))
    11. Function to get predictions
        List = []
        Best_Solution = Best_Population()
        For each observation in model_probability
            DotProduct = sigmoid of dot product of observation and Best_Solution
                If DotProduct >= 0.838
                    Append 1 into List
                Else
                    Append 0 into List
        Return List
    12. Function to get model Accuracy
        Input: TestDataY, predictions
        Correct = 0
        For y = 0 to length of TestDataY
            If TestDataY[y] = predictions[y]
                Correct += 1
        Return (Correct / length of TestDataY) * 100
End of the Class
```

The Differential Evolution method searches the search space that is defined by the lower and the upper bound values of each feature of the dataset, and thereafter generates new members of the population. A For-loop was created to loop through the algorithm and find the best performing number of population members. Another loop was created for learning the search space. The mutation method used was (X1 + Scaling factor * (X2 – X3)), where X1, X2 and X3 are the population members and Scaling factor is a random number between 0 and 2. The crossover method was performed on a mutation vector and the given population vector, and the cost was calculated.

A vector that achieves minimum cost was chosen as an optimum solution for the problem. An optimum solution vector is a set of weights. A set of weights were multiplied with predicted class probabilities, and a logistic function was used. The results of all algorithms defined in this chapter are analysed in detail in the next chapter. The next chapter discusses the data analysis.

4.8. Summary

This chapter discussed the algorithms that were used to carry out the research study. Various machine learning algorithms were defined and optimised to fulfil the research

objectives of this research study. The findings are explained in chapter 5 and 6 of this research study.

5. Chapter 5: Data Analysis

This chapter discusses the findings of this study. The overall objective of this research study is to develop an optimized credit card fraud detection model that can efficiently and effectively detect fraudulent transactions within a minimum number of attempts or the first time the fraudster attempts to commit fraud. Simply put, the model must overcome the weaknesses of the classifier models which fraudsters take advantage of in order to commit credit card fraud.

To satisfy the research objective, the base models of this study are executed and the output of each base model is observed. The output of each base model involves the overall accuracy, the predicted output values, and the probabilities of the predicted output values.

The data dictionary named Model_predictions defined in chapter 4 was used to store the predicted output values of all base models of the stacking ensemble method, and the data dictionary named Model_Probability defined in chapter 4 was used to store the probabilities of the predicted output values. The accuracy of base models, Model_predictions and Model_Probability with optimized weights of Differential Evolution were compared to evaluate the overall performance.

The violation of SMOTE and SL-SMOTE oversampling method definitions was noted in section 4.2. The SMOTE and SL-SMOTE are oversampling methods for balancing the class distribution of the dataset. The study seeks to demonstrate the claim that new data instance is not always generated between two existing data instances of the minority class of SMOTE. The demonstration of SMOTE and SL-SMOTE, and the modification of SMOTE and SL-SMOTE algorithms to always meet the main objective is shown in the section that follows.

5.1. SMOTE vs Safe-Level-SMOTE

This section of the research study seeks to demonstrate that SMOTE and SL-SMOTE algorithms do not always generate a new data instance between two existing data instances. In this study, the SMOTE and SL-SMOTE algorithms were modified such

that they always meet the main research objective defined in section 4.2. For demonstration purposes, the new random values of two data instances were generated.

The following random values of instances were chosen to test the claim made in this study: instances = [[-10, -21, -4, 45, -66, -93, 1], [10, 21, 4, 45, 66, 93, 1]]. For a pictorial display of the output, the x-axis values were required. The standard numbering system from 1 to 7 was used for visualization and demonstration purposes. The SMOTE and the modified SMOTE are demonstrated in the following sub-section.

5.1.1. SMOTE vs New SMOTE

The values of instances and standard numbering system were used in this sub-section to demonstrate the SMOTE and the modified SMOTE. The different colours were used to differentiate between the randomly generated values of instances and the new generated instance. The blue data points are randomly generated instances and the red data points are new data instance, respectively. The blue and red colouring system of the data points is adopted throughout the dissertation for the scatter plot. The pictorial representation of SMOTE method is shown in Figure 5.1.

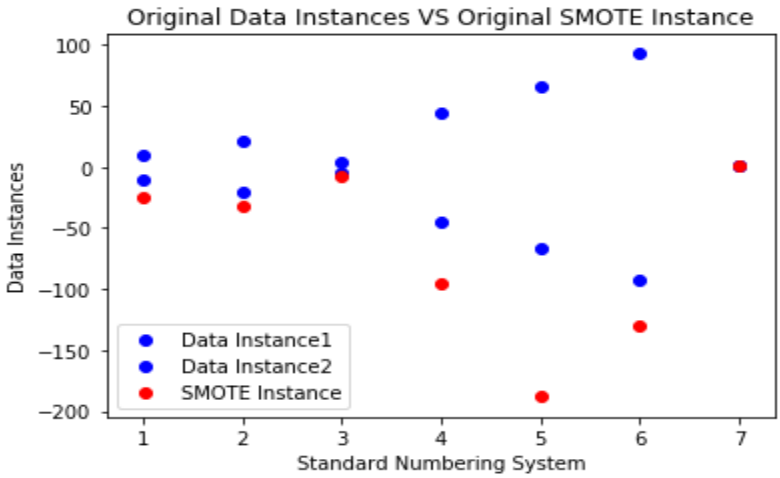


Figure 5.1: Original Data Instances VS Original SMOTE Instances.

According to Figure 5.1, the SMOTE instance was not generated between data instance1 and data instance 2. This means that the claim made in this research study about the SMOTE method is correct. The larger the space between the values of data instance 2 and data instance 1, the larger the space between generated data value and the values of data instance 2 and data instance 1.

To solve the problem of the SMOTE instances that are generated outside the boundaries of data instances 1 and 2, the **difference** variable in the attribute loop of SMOTE method outlined in section 4.2 must be modified to handle the issue of opposite signs. To this end, the Modified SMOTE algorithm outlined in section 4.2 is defined to handle the issue of opposite signs. A pictorial representation of the Modified SMOTE method is shown in Figure 5.2.

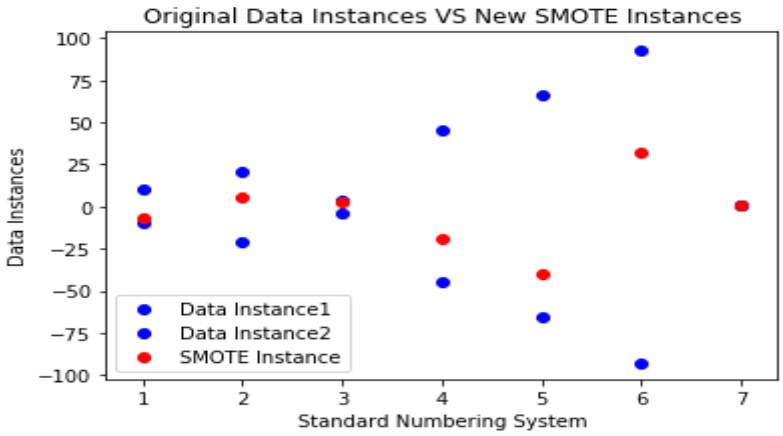


Figure 5.2: Original Data Instances VS New SMOTE Instances.

With respect to Figure 5.2, the SMOTE instance was generated within the boundaries of data instances 1 and 2. This suggests that the claim made about the SMOTE method in this research study is absolutely correct. A demonstration of the Safe-Level SMOTE and New SL-SMOTE algorithms is discussed in the following sub-section.

5.1.2. SL-SMOTE vs New SL-SMOTE

In this sub-section, the random values of instances and standard numbering system were also used to demonstrate the SL-SMOTE and the modified SL-SMOTE algorithms.

The same testing method used for SMOTE algorithm was used for the Safe_Level SMOTE algorithm because the manner in which a new data point is generated for the SMOTE and Safe_Level SMOTE algorithms is the same. Consequently, the visual depictions of both the SMOTE and the Safe_Level SMOTE algorithms are the same. The Safe-Level SMOTE method is presented pictorially in Figure 5.3.

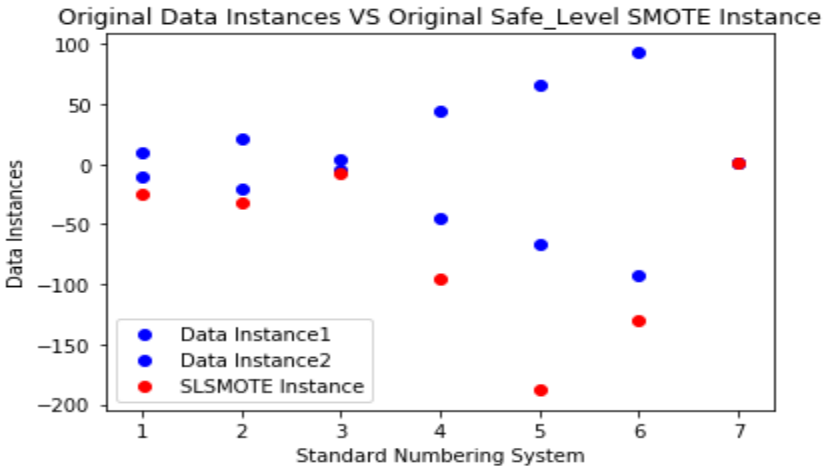


Figure 5.3: Original Data Instances VS Original Safe-Level SMOTE Instances.

The Safe-Level-SMOTE instance was generated outside the boundaries of data instances 1 and 2 in Figure 5.3 thus suggesting the correctness of the claim made about the Safe-Level SMOTE method.

To solve the problem of the Safe-Level SMOTE instance that are generated outside the boundaries of data instances 1 and 2, the **difference** variable in the attribute loop of Safe-Level SMOTE method listed in section 4.2 had to be modified to handle the opposite signs factor. In section 4.2, the Modified Safe-Level SMOTE algorithm is defined to handle the issue of opposite signs.

The same testing method used for the New SMOTE algorithm was also used for the New Safe_Level SMOTE because the manner in which a new data point is generated for the algorithm is the same for both algorithms. To this end, a visual depiction of the two algorithms (i.e. New SMOTE and New Safe_Level SMOTE) appears to be similar. As stated before, the blue data points are randomly generated instances, and the red data points are new data instance. A pictorial representation of Modified Safe-Level SMOTE method is shown in Figure 5.4.

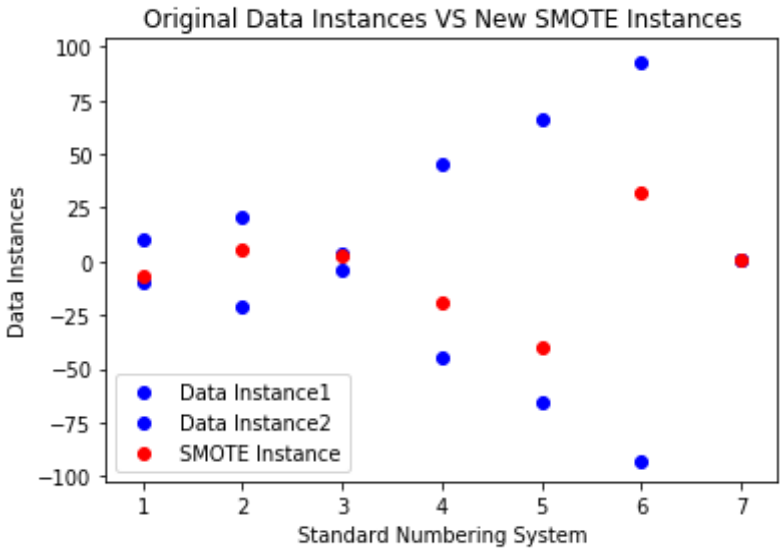


Figure 5.4: Original Data Instances VS Original New Safe-Level SMOTE Instances.

According to Figure 5.4, the SL-SMOTE instance was generated within the boundaries of data instances 1 and 2 thus suggesting that the claim made in this study about the SL-SMOTE method being correct.

The performance of the SMOTE, SL-SMOTE, modified SMOTE, and the modified SL-SMOTE methods were tested by computing the ANN, SVM, NB, DT and KNN algorithms defined in section 4.4. The four SMOTE methods were carried out in combination with the down-sampling of majority class data instances by removing duplicate data instances and by performing random sampling (see section 4.2). A grouped bar chart showing the performance of SMOTE algorithms against the chosen machine learning algorithms is shown in Figure 5.5.

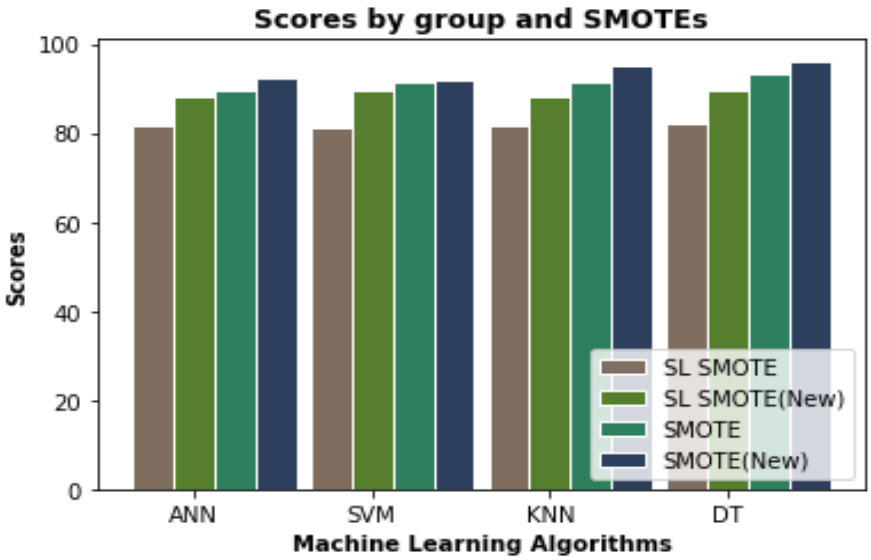


Figure 5.5: Scores by SMOTE groups

Although it is evident from Figure 5.5 that both the SMOTE and the Safe-Level SMOTE algorithms perform well. The two algorithms do not always generate a new data instance between two given data instances. This means that the two algorithms violate the objective of the smote algorithm. In this research study, the SMOTE and the Safe-Level SMOTE algorithms were re-designed such that they always met the objectives of the SMOTE algorithm while at the same time preserving and improving the capabilities of the algorithms. The re-designed Safe-Level SMOTE algorithm was therefore used to fulfil the objectives of this research study.

As discussed in the methodology chapter, the last step of the data engineering process involved balancing of the class distribution. The data engineering was followed by the data modelling, which was followed by defining the class objects of stacking ensemble base models defined in section 4.4. In this section that follows, the stacking ensemble class objects of the base model as well as the hyperparameters that are passed to the class definition of the base models during the creation of class objects are defined.

5.2. Stacking Ensemble Base Models

In this section, the class objects and the hyper-parameter values that are passed to class definition of the base models during the creation class objects are discussed. The hyper-parameter values are required for running the base model algorithms. Although, optimum parameter values are important for achieving the optimum

performance of the algorithms, the first base model for defining its class object and the hyper-parameter values involves the ANN method.

5.2.1. Artificial Neural Network (ANN) Method

To run the class code of ANN method defined in chapter 4, an object that contains class definition must be defined. The class object of ANN method was defined as nn, and nn contains the set of hyperparameter values below, which are passed to the class definition. A nested loop that loops through the number of hidden units and epochs is used to obtain optimum hyperparameter values. The code shown below illustrates the manner in which hyper-parameter values are passed to the class definition in order to create a class object.

```
N_CLASSES = 2
N_FEATURES = 29
RANDOM_SEED = 25
nn = NNClassifier(n_classes=N_CLASSES,
                  n_features=N_FEATURES,
                  n_hidden_units=123,
                  l2=0.5,
                  epochs=199,
                  learning_rate=0.001,
                  n_batches=25,
                  random_seed=RANDOM_SEED)
```

The second base model to define its class object and the hyperparameter values is the SVM method. The class object and the hyper-parameter values of the SVM method are defined in the sub-section that follows.

5.2.2. Support Vector Machine (SVM) Method

To run the class code of SVM method defined in chapter 4, an object that contains class definition must be defined. The object of SVM method in this study is defined as nnSVM, and nnSVM contains the set of hyperparameter values below are passed to the class definition. A loop to loops through the number of epochs was used to search for optimum number of epochs. The code shown below shows the manner in which the set of hyperparameter values are passed to the class.

```
N_CLASSES = 2
N_FEATURES = 29
RANDOM_SEED = 25
nnSVM = Support_Vaector_Machine(n_classes=N_CLASSES,
                  n_features=N_FEATURES,
```

```
                    learning_rate=0.001,
                    epochs = 300,
                    random_seed=RANDOM_SEED)
```

The next base model to define its class object and the hyperparameter values is the KNN method. The class object and the hyper-parameter values of the KNN method are defined in the subsection that follows.

5.2.3. k-Nearest Neighbour (KNN) Method

To run the class code of KNN method defined in chapter 4, an object that contains class definition must be defined. In this research study, the object of the KNN method was defined as nnKNN. nnKNN contains the following set of hyper-parameters, which are passed to the class definition. The code shown below shows the manner in which the set of hyper-parameter values that are passed to the class.

```
N_FEATURES = 29
N_NEIGHBORS = 5
TRAINx = Xs
TRAINy = Ys
RANDOM_SEED = 25
nnKNN = k_Nearest_Neighbor(row_length=N_FEATURES,
                           neighbors = N_NEIGHBORS,
                           TrainDataX = TRAINx,
                           TrainDataY = TRAINy,
                           random_seed=RANDOM_SEED)
```

Another base model for defining its class object and the hyper-parameter values is the Naïve Bayesian method. The class object and the hyper-parameter values of the Naïve Bayesian method are defined in the following subsection.

5.2.4. Naïve Bayesian (NB) Method

To run the class code of NB method defined in chapter 4 of this study, an object that contains class definition must be defined. The object of Naïve Bayesian method in this study is defined as nnNB, and nnNB contains the following set of hyperparameter values which are passed to the class definition. The code below shows the manner in which set of hyper-parameters are passed to the class.

```
TRAINx = Xs
TRAINy = Ys
RANDOM_SEEDS = 25
nnNB = Naive_Bayesian(TRAINx, TRAINy, RANDOM_SEEDS)
```

The last base model for defining its class object and the hyperparameter values is the Decision Tree method. The class object and the hyperparameter values of Decision Tree method are defined in the subsection that follows.

5.2.5. Decision Tree (DT) Method

An object that contains class definition must be defined in order to run the class code of the DT method defined in chapter 4. The object of Decision Tree method in this research study is defined as nnDT, which contains the following set of hyper-parameters that are passed to the class definition. The Entropy was used as the criterion for training the tree. The optimum maximum depth and minimum sample leaf chosen were optimum. The code shown below illustrates the manner in which set of hyper-parameter values are passed to the class.

```
MAX_DEPTH = 3
MIN_SAMPLES_LEAF = 5
ENTROPY_CRITERION = 'entropy'
RANDOM_SEED = 1
nnDT = Decision_tree(Max_depth = MAX_DEPTH,
                     Min_samples_leaf = MIN_SAMPLES_LEAF,
                     Entropy_Criterion = ENTROPY_CRITERION,
                     random_seed = RANDOM_SEED)
```

The class objects were created by passing the hyper-parameter values that enables stacking ensemble base models to learn the dataset, and were used for accessing the defined class properties. After defining the class objects of stacking ensemble base models, the class objects were deemed ready to be used for accessing the defined class definitions. In the following section, the manner in which the defined performance class is accessed through the usage of class objects is described.

5.3. Performance of Base Models

The class objects in section 5.2 are created by passing the optimum hyper-parameter values to class definition of base models defined in section 4.4. The class objects of stacking ensemble base models described in this section were used to access the defined performance property of the base model classes in order to compare the performance.

The model's overall accuracy as well as the true negative transactions, true positive transactions, false negative transactions, and false positive transactions. The

accuracy of the base models, which involves the overall accuracy and confusion matrix of the model, are discussed in the section that follows.

5.3.1. Accuracy of Base Models

The python code below prints the name and the overall accuracy score of the base models. Nn, nnSVM, nnKNN, nnDT, and nnNB are the class objects defined in section 5.2, and score and getAccuracy are the class properties of base models in section 4.4. The X_test and y_test are the respective input and output values of the test data. The accuracy of the base model was achieved by running the following code:

```
print('Artificial Neural Network: '+str(nn.score(X_test, y_test)))
print('Support Vector Machine: '+str(nnSVM._score(X_test, y_test)))
print('K-Nearest Neighbor: '+str(nnKNN._getAccuracy(list(y_test),
knn_pred)))
print('Decision Tree: '+str(nnDT._getAccuracy(list(y_test), predProb)))
print('Naive Bayesian: '+str(nnNB._getAccuracy(list(y_test), Predicted_y)))

Output:

Artificial Neural Network: 0.9518275341455708
Support Vector Machine: 0.7108598714932762
K-Nearest Neighbor: 90.83482555621877
Decision Tree: 99.17488852217268
Naive Bayesian: 97.06002832297555
```

When compared with other base models, the Decision Tree Method performed extremely well and achieved a higher accuracy rate. The Support Vector Machine method achieved an accuracy that is far lower than that of other base models. The overall performance of all the base models is good. However, the statistics to show which class of the distribution is well classified is not shown.

5.3.2. Confusion Matrix

The confusion matrix is a classification model evaluation method that shows the total number of true negative, true positive, false negative, and false positive. The number of true positive (TP) transactions is the number of output transactions that the base classifier has predicted as being fraudulent when they were in fact fraudulent. Similarly, the count of true negative (TN) transactions is the count of output transactions that the base classifier has predicted as being non-fraudulent when they were in fact non-fraudulent, the count of false positive (FP) transactions is the count of output transactions that the base classifier has predicted as being fraudulent when they are non-fraudulent, and the count of false negative (FN) transactions is the count

of output transactions that the base classifier has predicted as being non-fraudulent when they are in fact fraudulent.

The function definition of the confusion matrix Method is defined in algorithm 18 of section 8.4. The following pseudo-code shows the flow of how the confusion matrix Method was defined in section 8.4.

Algorithm 18: Confusion Matrix Method

Input: PredictedClassValues, ClassValues
Output: Confusion Matrix Scores
Begin

```
Pivot = []
For each column in columns
    Column = column value
    Initial True Positive(TP), True Negative(TN), False Positive(FP) and False Negative(FN)
    For ind = 1 to length of the Column
        IF ClassValue and PredictedClassValue at ind are both equal to 1
            Then TP += 1
        ELIF ClassValue at ind is equal to 1 and PredictedClassValue at ind is equal to 0
            Then FN += 1
        ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 1
            Then FP += 1
        ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 0
            Then TN += 1
Append TP, TN, FP,FN into the Pivot
Return Pivot
```

End of the function

To run the confusion matrix, the following function object code was used: ConfusionMatrix(Model_predictions, y_test). The total number of illegal and legal transactions for testing dataset was 137 and 85306.

Table 5.1: Confusion matrix.

	Model	TP	TN	FP	FN
	ANN	5	81130	4176	132
	SVM	22	61753	23553	115
	KNN	7	77359	7947	130
	DT	0	84492	814	137

	Model	TP	TN	FP	FN
	NB	4	82699	2607	133

It is clear from negative (legal) and positive (illegal) transactions listed in table 5.1 that the base models are extremely biased. To give the above confusion matrix table meaning, the table can be shown in a form of statistics. The function definition of the confusion matrix stats Method is defined in algorithm 19 of section 8.4. The following pseudo-code shows the flow of how the confusion matrix Method was defined in section 8.4.

Algorithm 19: Confusion Matrix Stats Method

Input: PredictedClassValues, ClassValues
Output: Confusion Matrix Scores
Begin

```
Pivot = []
For each column in columns
    Column = column value
    Initial True Positive(TP), True Negative(TN), False Positive(FP) and False Negative(FN)
    For ind = 1 to length of the Column
        IF ClassValue and PredictedClassValue at ind are both equal to 1
            Then TP += 1
        ELIF ClassValue at ind is equal to 1 and PredictedClassValue at ind is equal to 0
            Then FN += 1
        ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 1
            Then FP += 1
        ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 0
            Then TN += 1
    Class0, Class1 = 0, 0
    For each value in the ClassValue
        IF value = 0
            Class0 += 1
        ELIF value = 1
            Class1 += 1
    Append TP/Class1, TN/Class0, FP/Class1,FN/Class0 into the Pivot
Return Pivot
```

End of the function

To run the confusion matrix statistics function, the following function object code was used: ConfusionMatrixStat(Model_predictions, y_test). The output of confusion matrix statistics is shown in Table 5.2.

Table 5.2: Confusion Matrix Statistics.

	Model	TP	TN	FP	FN
	ANN	0.036496	0.951047	30.481752	0.001547
	SVM	0.160584	0.723900	171.919708	0.001348
	KNN	0.051095	0.906841	58.007299	0.001524
	DT	0.000000	0.990458	5.941606	0.001606
	NB	0.029197	0.969439	19.029197	0.001559

It is clear from the negative (FN) transactions listed in table 5.2 that the Decision Tree method had a higher percentage of accuracy compared to other base models. However, the Decision Tree method was extremely biased compared to other base models; the Support Vector Machine method had a lower percentage of accuracy when compared with other base models. On the other hand, the true positive (TP) of Support Vector Machine method is greater than the true positive value of other base models.

Therefore, it can be concluded that a higher percentage of accuracy of the base model that is extremely biased is meaningless compared to an average percentage of accuracy of the base model where all output class values are well represented. In this study, the true accuracy of a base model is the accuracy in which all output classes are well represented.

The stacking ensemble base models were optimised to achieve equal representation of output classes while ensuring that the high overall accuracy of base models is maintained. The optimization methods of stacking ensemble base models are discussed in the next section.

5.4. Optimized Stacking Ensemble Base Models

To obtain the true accuracy of base models and thus achieve a classification problem of equal representation of class values, the hyper-parameters of the base models were modified. Furthermore, a classification problem of equal representation of class values was optimized to maintain the higher overall accuracy of base models. The definitions

of the base models (outlined in section 4.4) and class objects (outlined in section 5.3) were modified. A description of the manner in which each of the base models was modified in order to maintain a high level of is as follows

- **Optimization of the ANN model** – for the ANN base model, the number of epochs was manually changed from 199 to 50 as defined in section 5.1.1, and used a FOR loop to loop through the hidden layers. The best number of hidden layers in relation to the epochs was achieved by computing the following code.

```
ListItems = []
for ind in range(500):
    nn = NNClassifier(n_classes=N_CLASSES,
                      n_features=N_FEATURES,
                      n_hidden_units=ind,
                      l2=0.5,
                      epochs=50,
                      learning_rate=0.001,
                      n_batches=25,
                      random_seed=RANDOM_SEED)
    Ann_pred = nn.predict(X_test)
    CM = ConfusionMatrixDE(Ann_pred, y_test)
    if np.array(CM)[0][1]/137 >= 0.50 and np.array(CM)[0][2]/85306 >=
0.50:
        ListItems.append([ind, np.array(CM)[0][1]/137,
np.array(CM)[0][2]/85306])
```

  The ListItems contains a list of hidden layers, true positive and true negative statistics. The hidden layer highest performance and good balance of class representation statistics between illegal transactions and legal transactions were chosen. The number of nearest neighbours was also changed from 5 to 1 (see section 5.1.3).

- **Optimization of the SVM model** – In the case of the SVM model, the classifier value was changed from 0 to 0.1 as stipulated in section 4.4.2.
- **Optimization of the KNN model** – In the case of the KNN model, the number of nearest neighbors was changed from 5 to 1 as stipulated in section 5.1.3.
- **Optimization of the Decision Tree model** – for the Decision Tree model, the training method was changed from entropy criterion to Gini criterion in section 4.4.5.
- **Optimization of the Naïve Bayesian model** - the predict function of the Naïve Bayesian class in section 4.4.4 was modified by multiplying the class probability

of legal transactions by 0.51 and the class probability of illegal transaction by 0.41 for each transaction. The Naïve Bayesian prediction function was redefined as follows:

```
def _predict(self, TestDataInstance):
        probabilities =
self._calculateClassProbabilities(TestDataInstance)
        List, Clas = [], 0
        for classValue, probability in probabilities.items():
            List.append([classValue, probability])
        Prob0 = List[0][1] * 0.51
        Prob1 = List[1][1] * 0.49
        if Prob0 > Prob1:
            Clas = List[0][0]
        else:
            Clas = List[1][0]
        return Clas
```

The naïve Bayesian function is redefined inside the naïve Bayesian class.

Once all the base models were optimized, the class object were used in combination with the class properties to re-run the accuracy code and print the true accuracies of the base models. The true accuracies of the base models are detailed in the following sub-section.

5.4.1. True Accuracy of Base Models

The python code shown below was used to prints the name and the overall true accuracy score of the base models. The class object and the class properties were used to run the accuracy code to print the true accuracies of the base models. Whereas the X_test refers to the input values of the test data, the y_test refers to the output values of the test data. The true accuracy of the base model was achieved by running the following code:

```
print('Artificial Neural Network: '+str(getAccuracy(Ann_pred,
np.array(y_test))))
print('Support Vector Machine: '+str(getAccuracy(SVM_pred,
np.array(y_test))))
print('K-Nearest Neighbor: '+str(getAccuracy(knn_pred, np.array(y_test))))
print('Decision Tree: '+str(getAccuracy(DT_pred1, np.array(y_test))))
print('Naive Bayesian: '+str(nnNB._getAccuracy(list(y_test), NB_pred)))

Output:

Artificial Neural Network: 95.19562749435296
Support Vector Machine: 72.8637805320506
K-Nearest Neighbor: 92.20650023992604
Decision Tree: 95.18158304366654
```

```
Naive Bayesian: 96.94065049214096
```

The Naïve Bayesian method performed extremely well; relative to other base models, a higher accuracy rate was achieved. The accuracy of the SVM method was far lower than those of other base models. To check how well the class values of base models are represented, the confusion matrix is required. The true confusion matrix is discussed in the sub-section that follows.

5.4.2. True Confusion Matrix

This shows the number of true negative, true positive, false negative, and false positive transactions, which are defined in section 5.3.2. The confusion matrix function was used to generate the following confusion matrix shown in Table defined in section 5.3.2 of this study. For the testing of the dataset, the total count of illegal transactions and the total count of legal transactions of 137 and 85306 were used, respectively.

Table 5.3: True Confusion Matrix.

	Model	TP	TN	FP	FN
	ANN	98	81240	4066	39
	SVM	136	62121	23185	1
	KNN	126	78658	6648	11
	DT	125	84711	595	12
	NB	124	82705	2601	13

To give the above confusion matrix more meaning, the confusion matrix statistics is required. To display the confusion matrix statistics, the confusion matrix statistics function defined in section 5.3.2 was used. The output of the confusion matrix statistics is shown in Table 5.4.

Table 5.4: True Confusion Matrix Statistics.

	Model	TP	TN	FP	FN
	ANN	0.715328	0.952336	29.678832	0.000457
	SVM	0.992701	0.728214	169.233577	0.000012
	KNN	0.919708	0.922069	48.525547	0.000129
	DT	0.912409	0.993025	4.343066	0.000141
	NB	0.905109	0.969510	18.985401	0.000152

The confusion matrix statistics in Table 5.4 shows that the true positive and negative transactions are well represented. This means that the fraudulent data and non-fraudulent data instances are well classified. Since the true accuracy of a base model is the accuracy in which all output classes are well represented, it can be concluded that the accuracies in section 5.4.2 are true accuracies of base models. The true accuracies are required to model a stacking ensemble method. Stacking ensemble method is a classification method that uses more than one classification model for decision making, and it is discussed in the next section.

5.5. Stacking Ensemble Method

The Stacking Ensemble Method is the final voting of the class values generated by the base modes defined in section 4.4. The voting class named Voting, which is discussed in section 4.6, allows base models to vote the final output of each transaction of credit card data. To run the Voting class code, the class Voting() was used as an object. The class object contains the class definition called class properties. The Model_predictions data dictionary that contains predicted class values by the base models was used for voting of the final class values. The following class object was used to return a list of predicted output values:

```
Final_pred = Voting().Vote_Func(np.array(pd.DataFrame(Model_predictions)))
```

Vote_Func is the class property of Voting class, which returns the list of final prediction of class values that belongs to credit card transactions. The list of predicted class values was required to compute the accuracy of stacking ensemble method. The

accuracy of the stacking ensemble method that represents the class values of the dataset well while maintaining or enhancing the performance of the classification model is essential to gain a confidence in the classifier model. The accuracy of the stacking ensemble method is discussed in the next sub-section.

5.5.1. Stacking Ensemble Method Accuracy

The stacking ensemble method accuracy was the accuracy that represents the class values of the dataset well while maintaining or enhancing the performance of the classifier model. The voting class property of accuracy was getAccuracy which takes two parameters, namely: the test output values of the dataset named y_test, and the predicted output of credit card transactions named Final_pred. To display the accuracy of the stacking ensemble method, the following python code was executed: `Voting().getAccuracy(np.array(y_test), Final_pred).`

The output of 99.26500708074389 suggested that the stacking ensemble method was performing well. However, to determine how well the stacking ensemble method was performing, the confusion matrix defined in sub-section 5.3.2 was required. The confusion matrix of stacking ensemble method is discussed in the subsection that follows.

5.5.2. Confusion Matrix of Stacking Ensemble.

This shows the number of true negative, true positive, false negative, and false positive transactions. The confusion matrix was discussed in more detail in section 5.3.2. The function definition of the confusion matrix of Stacking Ensemble Method is defined in algorithm 20 of section 8.4. The following pseudo-code shows the flow of how the confusion matrix Method was defined in section 8.4.

Algorithm 20: Confusion Matrix for Stacking Ensemble Method

Input: PredictedClassValues, ClassValues
Output: Confusion Matrix Scores
Begin

```
1. Pivot = []
2. For each column in columns
3.     Column = column value
4.     Initial True Positive(TP), True Negative(TN), False Positive(FP) and False Negative(FN)
5.     For ind = 1 to length of the Column
6.         IF ClassValue and PredictedClassValue at ind are both equal to 1
7.             Then TP += 1
```

```
        ELIF ClassValue at ind is equal to 1 and PredictedClassValue at ind is equal to 0
            Then FN += 1
        ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 1
            Then FP += 1
        ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 0
            Then TN += 1
    Append TP, TN, FP,FN into the Pivot
Return Pivot
End of the function
```

The Model_predictions is the data dictionary that contains predicted class values of base models that represent the class values well, and y-test is the output values of the testing dataset. Vote_Func is the class property of Voting class that returns the list of the final prediction of class values belonging to credit card transactions. The above confusion matrix function was executed, and the output is given in Table 5.5:

Table 5.5: Confusion Matrix of Stacking Ensemble.

Model	TP	TN	FP	FN
Stacking Ensemble	1	84814	478	150

The total number of illegal and legal transactions is 137 and 85306, respectively. The output of stacking ensemble method confusion matrix shows that the both base models and stacking ensemble method are performing well. However, to enhance the understanding of the confusion matrix, the statistics of the confusion matrix was required. The statistics of the stacking ensemble method confusion matrix is discussed in the following sub-section.

5.5.3. Confusion Matrix Statistics of Stacking Ensemble.

The statistics function of the stacking ensemble method confusion matrix was designed to show how well the class values are represented (in a form of percentages) by the prediction model. The function definition of the confusion matrix stats of Stacking Ensemble Method is defined in algorithm 21 of section 8.4. The following pseudo-code shows the flow of how the confusion matrix Method was defined in section 8.4.

Algorithm 21: Confusion Matrix Stats for Stacking Ensemble Method

```
Input: PredictedClassValues, ClassValues
Output: Confusion Matrix Scores
Begin
    Pivot = []
    For each column in columns
        Column = column value
        Initial True Positive(TP), True Negative(TN), False Positive(FP) and False Negative(FN)
        For ind = 1 to length of the Column
            IF ClassValue and PredictedClassValue at ind are both equal to 1
                Then TP += 1
            ELIF ClassValue at ind is equal to 1 and PredictedClassValue at ind is equal to 0
                Then FN += 1
            ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 1
                Then FP += 1
            ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 0
                 Then TN += 1
        Class0, Class1 = 0, 0
        For each value in the ClassValue
             IF value = 0
                 Class0 += 1
             ELIF value = 1
                 Class1 += 1
    Append TP/Class1, TN/Class0, FP/Class1,FN/Class0 into the Pivot
    Return Pivot
End of the function
```

Whereas the Model_predictions is the data dictionary that contains predicted class values of base models representing the class values well, and y-test is the output values of the testing data. The above confusion matrix code was executed, and the output is displayed in Table 5.6.

Table 5.6: Confusion Matrix Statistics of Stacking Ensemble.

Model	TP	TN	FP	FN
Stacking Ensemble	0.006623	0.994396	3.165563	0.001759

While the total number of illegal transactions was found to be 137, and the total number of legal transactions was 85306. The statistics of the stacking ensemble method confusion matrix shows how well the class values are represented by the prediction model. However, a good classification model minimises the count of false negatives and positives, even when the count of true negatives and positives is good in relation to the total count of transactions.

Therefore, this research study sought to optimize the stacking ensemble method to decrease the number of misclassified transaction while enhancing the performance of

the stacking ensemble method. The optimization technique that was adopted for this research study is Differential Evolution Method, which is detailed in the next sub-section.

5.6. Differential Evolution Method

To run the class code of Differential Evolution Method defined in section 4.8, an object containing class definition must be defined. The object of the Differential Evolution Method is defined as DE, and DE contains the set of parameter values of the class that are passed to the constructor. The code shown below shows how the set of hyper-parameters that are passed to the class and the Model_proba_df1, which is the Data-Frame of predicted probabilities of transactions representing the class values well.

```
TrainDataX = Model_proba_df1
population_size = 100000
Scaling_Factor = 0.9
Max_Iterator = 500
Crossover_Probability = 1.2
random_seed = 25

DE = DifferentialEvolution(TrainDataX,
                           population_size,
                           Scaling_Factor,
                           Max_Iterator,
                           Crossover_Probability,
                           random_seed)
```

The object code DE.Best_Population() was used to return the error rate named “Best Generation” and the best weight solution named “Best Solution”. The output of `DE.Best_Population()` is as follows:

```
Best Generation: 0.9784773967015673
Best Solution: [0.5191182431959512, -0.21487774856187453, 0.600552155776322
6, 0.5496892828208568, 1.384309253288074e-11]
```

The best solution is the set of weights which are multiplied by the predicted class pro babilities of each transaction. The logistic function and the optimized threshold of 0.8 38 were used for differential evolution decision making. The accuracy of differential e volution model was required to determine how the model was performing. Differential evolution optimization was performing well if the class values of transactions were w ell represented and high accuracy of the model was preserved. The accuracy of the differential evolution is discussed in the next sub-section.

5.6.1. Differential Evolution Accuracy

The differential evolution accuracy refers to the accuracy in which the class values of differential evolution prediction model are well represented while maintaining or enhancing the performance of the model. The accuracy in which the class values of the classification model are well represented is defined as true accuracy of the model. To perform the true accuracy of differential evolution model, the class object was required.

In this research study, the object of the Differential Evolution Method was defined in section 5.4 as DE. While Model_Probability_df is the data-frame that contains the predicted class probability of credit card transactions, getPredictions is the function of differential evolution that returns the list of predicted output classes of credit card transactions. On the other hand, getAccuracy is also the function of differential evolution that returns the accuracy of the differential evolution technique. The accuracy of Differential Evolution Model was achieved by running the following code:

```
predictions = DE.getPredictions(Model_proba_df1)
DE.getAccuracy(np.array(y_test), predictions)
```

y_test is the output values of the test data described in the data split function in section 4.1. The true accuracy of Differential Evolution Optimization Model is 99.9133925541004. To check how well the class values of Differential Evolution Optimization Model were represented, confusion matrix was required, and it is outlined in the following sub-section.

5.6.2. Differential Evolution Confusion Matrix

This shows the number of true negative, true positive, false negative, and false positive transactions. The confusion matrix was discussed in great detail in section 5.3.2. The function definition of the differential evolution confusion matrix Method is defined in algorithm 22 of section 8.4. The following pseudo-code shows the flow of how the confusion matrix Method was defined in section 8.4.

Algorithm 22: Confusion Matrix for Differential Evolution Method

Input: PredictedClassValues, ClassValues
Output: Confusion Matrix Scores
Begin

1. Pivot = []
2. For each column in columns
3. Column = column value
4. Initial True Positive(TP), True Negative(TN), False Positive(FP) and False Negative(FN)
5. For ind = 1 to length of the Column
6. IF ClassValue and PredictedClassValue at ind are both equal to 1
7. Then TP += 1
8. ELIF ClassValue at ind is equal to 1 and PredictedClassValue at ind is equal to 0
9. Then FN += 1
10. ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 1
11. Then FP += 1
12. ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 0
13. Then TN += 1
14. Append TP, TN, FP,FN into the Pivot
15. **Return** Pivot

End of the function

The Model_proba_df1 refers to the Data-Frame of predicted probabilities of transactions that represent the class values well. The getPredictions is the property of DE class object and is used to return the list of predicted class values of credit card transactions. The Model_proba_df1 Data-Frame, the class object, the getPredictions class property, and the confusion matrix function shown below were used to display the differential evolution confusion matrix indicated in Table 5.7.

```
predictions = DE.getPredictions(Model_proba_df1)
ConfusionMatrixDE(predictions, y_test)
```

Table 5.7: Differential Evolution Confusion Matrix.

Model	TP	TN	FP	FN
Differential Evolution	117	85252	54	20

y_test is the output values of the test data described in section 4.1 under the data split function. The total count of illegal and legal transactions is 137 and 85306, respectively. The output Differential Evolution confusion matrix shows that both the base model and Differential Evolution Optimization Model are performing well. The class values of the prediction models are well represented and the overall accuracy is good. To enhance the meaning of the above confusion matrix, the statistics of confusion matrix was required. For this reason, the statistics of the Differential Evolution confusion matrix is discussed in the following sub-section.

5.6.3. Differential Evolution Confusion Matrix Statistics

In this research study, the differential evolution confusion matrix statistics function was designed to establish to how well the class values are represented (in a form of percentages) by the prediction model. The function definition of the differential evolution confusion matrix statistics Method is defined in algorithm 23 of section 8.4. The following pseudo-code shows the flow of how the confusion matrix Method was defined in section 8.4.

Algorithm 23: Confusion Matrix Stats for Differential Evolution Method

Input: PredictedClassValues, ClassValues
Output: Confusion Matrix Scores
Begin

```
Pivot = []
For each column in columns
    Column = column value
    Initial True Positive(TP), True Negative(TN), False Positive(FP) and False Negative(FN)
    For ind = 1 to length of the Column
        IF ClassValue and PredictedClassValue at ind are both equal to 1
            Then TP += 1
        ELIF ClassValue at ind is equal to 1 and PredictedClassValue at ind is equal to 0
            Then FN += 1
        ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 1
            Then FP += 1
        ELIF ClassValue at ind is equal to 0 and PredictedClassValue at ind is equal to 0
            Then TN += 1
    Class0, Class1 = 0, 0
    For each value in the ClassValue
        IF value = 0
            Class0 += 1
        ELIF value = 1
            Class1 += 1
Append TP/Class1, TN/Class0, FP/Class1,FN/Class0 into the Pivot
Return Pivot
```

End of the function

The Data-Frame of predicted probabilities of transactions that represent the class values well is referred to as the Model_proba_df1. The getPredictions of DE class object was used to return the list of predicted class values of credit card transactions. The Model_proba_df1 Data-Frame, the class object, the getPredictions class property, and the confusion matrix statistics function shown below were used for displaying the Differential Evolution confusion matrix.

```
predictions = DE.getPredictions(Model_proba_df1)
ConfusionMatrixStatDE(predictions, y_test)
```

Table 5.8: Differential Evolution Confusion Matrix Statistics.

Model	TP	TN	FP	FN
Differential Evolution	0.854015	0.999367	0.394161	0.000234

The output values of the test data (i.e. y_test) was defined in the data split function described in section 4.1; the total count of illegal transactions is 137 and the total count of legal transactions is 85306. The output of true differential evolution confusion matrix statistics shows how well the class values are represented by the prediction model. In the next subsection, a comparative analysis of the confusion matrix of the Differential Evolution Optimization Method and the Stacking Ensemble Method is undertaken.

5.7. Differential Evolution Method vs Stacking Ensemble Method

The confusion matrix is explained in details in sub-section 5.3.2. Figure 5.6 shows the true negative and true positive transactions of the stacking ensemble method and the classification problem of the differential evolution optimization of stacking ensemble method.

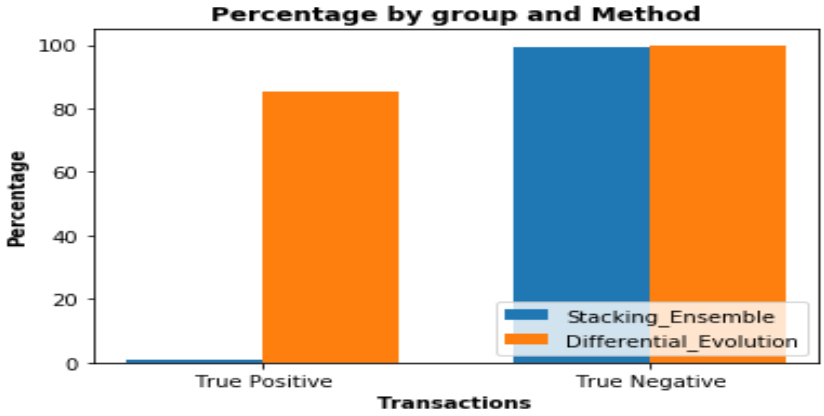


Figure 5.6: True Positive/Negatives of Stacking Ensemble vs Differential Evolution Methods.

It is clear from Figure 5.6 that the number of true positive transactions (fraudulent transactions) generated from the stacking ensemble method is exceeded by that of the true positives generated using the Differential Evolution Optimization of Stacking Ensemble method. Put differently, the number fraudulent transaction detected by the differential evolution optimization of Stacking Ensemble classifier is substantially higher than those of Stacking Ensemble model.

The number of true negative transactions (non-fraudulent transactions) of the Stacking Ensemble method is lower than that of the true negatives of the differential evolution optimization of stacking ensemble method. This suggest that the number of non-fraudulent transaction detected using the Differential Evolution Optimization of Stacking Ensemble classifier are greater than those detected by the Stacking Ensemble method.

A comparative analysis of the misclassified transactions using the differential evolution optimization of stacking ensemble classifier and the stacking ensemble classifier is displayed in Figure 5.7.

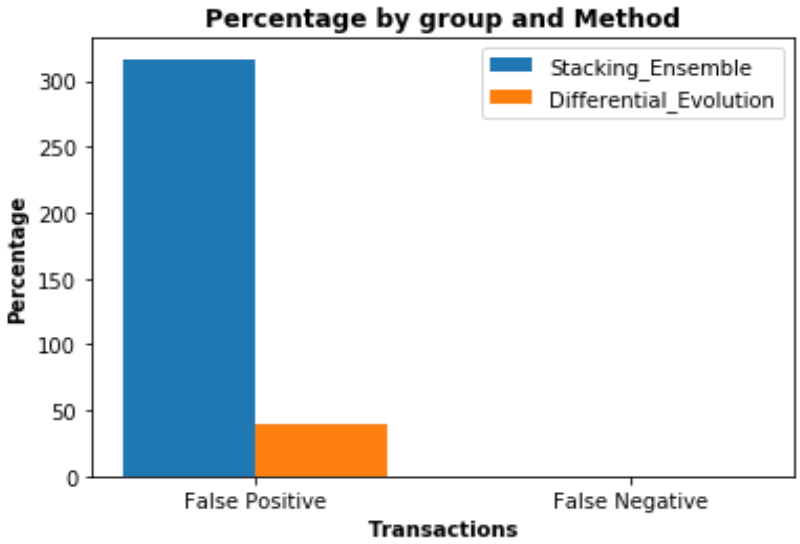


Figure 5.7: False Positive/Negatives of Stacking Ensemble vs Differential Evolution Methods.

According Figure 5.7, the number of false positive transactions (misclassified fraudulent transactions) generated using the Stacking Ensemble method is slightly higher than the false positives associated with the Differential Evolution Optimization of Stacking Ensemble method. Relative to the false positive transactions, the percentage of negative transactions (misclassified non-fraudulent transactions) is significantly low and therefore not visible.

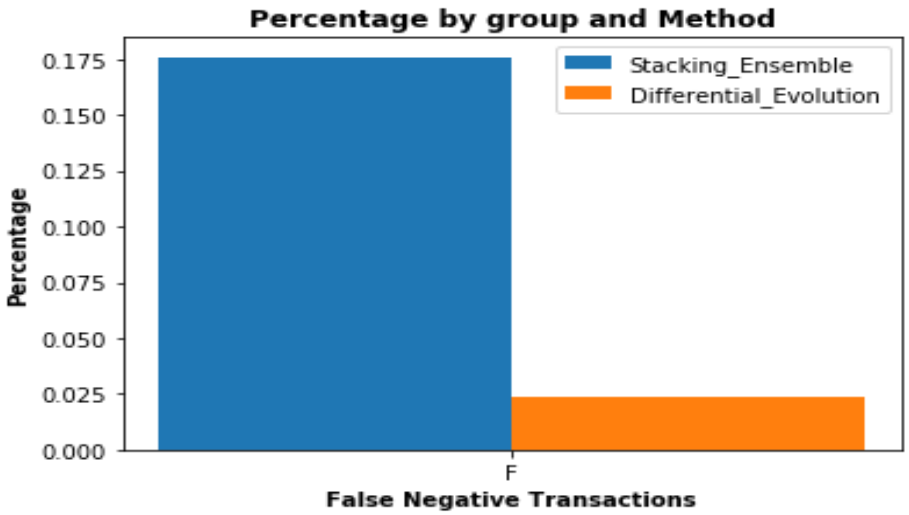


Figure 5.8: False Negatives of Stacking Ensemble vs Differential Evolution Methods.

Figure 5.8 shows the number of false negative transactions (misclassified non-fraudulent transactions) for the two methods. The number of false negative transactions generated through the stacking ensemble method is greater than that of the Differential Evolution Optimization of Stacking Ensemble method.

5.8. Summary

In this chapter, the presentation, the analysis and interpretation of the results of the research study were put forward. It is found that the differential evolution optimization of stacking ensemble classifier outperforms the stacking ensemble classifier in terms of classification and misclassification of credit card fraudulent transactions. The discussion and conclusion of the findings are explained in chapter 6 of this research study.

6. Chapter 6: Discussion and Conclusion

This last and final chapter of this research study is composed of two sections. Efforts in the first section are aimed at establishing whether the research questions that were raised in the first chapter have been addressed. The second section concludes this research study and recommends the future works to address the limitations of the study.

6.1. Discussion

The **Main Research Question**, which was outlined in Chapter 1, was:

> *"Is the Differential Evolution optimization method a good approach for defining the weighting function to predict the outcome of future credit card fraudulent transactions?"*

As indicated in Chapter 5 (Data Analysis), the differential evolution stochastic optimization method shows that the weighted voting stacking ensemble method, when used in combination with a differential evolution stochastic optimization method performed better than the base models.

When used in combination with the differential evolution stochastic optimization method, the weighted voting stacking ensemble method was able to achieve higher accuracy for confusion matrix true negative (TN) and true positive (TP) transactions. The confusion matrix false negative (FN) and false positive (FP) transactions of the credit card fraud data were, on the other hand, extremely low.

This means that when used in combination with the differential evolution stochastic optimization method, the weighted voting stacking ensemble method performed very well without any biasness of the class distribution. Thus, the differential evolution optimization method is a good approach for defining the weighting function to predict the outcome of future fraudulent credit card transactions.

The following sub-research questions for addressing the problem statement were asked:

(a) First Sub-research Question

"Does Safe-Level-SMOTE oversampling method (on the minority classes) used with under-sampling method (that eliminates duplicate data samples on the majority class) have positive impact on reducing the high skewedness of the class distribution than SMOTE oversampling method (on the minority classes) used with the under-sampling method (that also eliminates duplicate data samples on the class with majority data observations)?"

The SMOTE and the Safe-Level-SMOTE methods were computed with under-sampling methods that eliminates duplicate data observations and randomly select the data observations of the class with majority data observations to balance the class distribution. The SL-SMOTE method actually performed better than SMOTE. However, the logic behind the mathematical representation of both the SMOTE and the SL-SMOTE methods was found to be violating the objective of SMOTE method. For this reason, the modified SMOTE and the modified SL-SMOTE methods were designed and tested in this research study.

The modified SMOTE method was found to perform better than the modified SL-SMOTE method when tested with the credit card dataset used in this research study. Lastly, it can be concluded that both SMOTE and SL-SMOTE methods are powerful oversampling methods when used in combination with majority class data observations that were down-sampled by removing duplicate data observations and randomly select the data observation to balance the class distribution. Therefore, it can be concluded that incorrect computation of an algorithm can cloud the true computing capabilities of an algorithm.

(b) Second Sub-research Question

"Does Safe-Level-SMOTE oversampling method and the under-sampling method (that eliminates duplicate data samples on the class of the majority data observations) reduce or eliminate the problem of overlapping data samples between fraudulent and non-fraudulent classes?"

The Safe-Level-SMOTE oversampling method and the under-sampling method that eliminates duplicate data samples and randomly select data observations from the majority class to balance the class distribution did not eliminate the overlapping data observations since the training dataset was not entirely separable.

The training dataset was not entirely separable because the SVM that is known to be wide road separator algorithm did not achieve the 100% accuracy. That is to say, the SVM method is the wide separator of illegal and legal transactions, and its inability to separate the two transactions speaks volumes about the dataset. However, it can be concluded that the overlapping data observations were reduced since all the base models (machine learning algorithms) performed well to classify the illegal and legal transactions according to their respective classes.

(c) Third Sub-research Question

"Does manual stacking of feature selection methods that are diverse on the PCA transformed features to hide the actual feature names, select enough good features to solve the problem than just selecting a feature selection algorithm?"

The manual stacking of feature selection methods that are diverse on the PCA transformed features performed differently for each of the features. However, similarities were noted for features that were not adding or adding less value to the classification problem. The time feature appeared to be adding less value to the classification problem when computing univariate selection algorithm, recursive feature elimination algorithm, and feature importance algorithm. For this reason, the time feature was removed from the dataset.

(d) Fourth Sub-research Question

“Does the stacking of the supervised machine learning algorithms have an impact on predictive accuracy given the under-sampling technique to remove the duplicate data observations on the majority class, and the Safe-Level-SMOTE oversampling method?”

The stacking of the supervised machine learning algorithms has proven to positively impact the predictive accuracy given the under-sampling technique to remove the duplicate data observations on the majority class, and the Safe-Level-SMOTE oversampling method. Like the base models (machine learning methods), the stacking method was found to be extremely biased.

However, the stacking method had fewer false negatives and false positives compared to the base models. This means that the stacking ensemble method performed slightly better than the base models. Therefore, the claim about the majority voting stacking ensemble method not being a good stacking ensemble method for the credit card dataset as compared to the weighted voting stacking ensemble method was correct.

(e) Fifth Sub-research Question

“Will the proposed data model for fraud detection efficiently detect credit card fraudulent activities?”

The proposed method for fraud detection is the weighted voting stacking ensemble method used in combination with the differential evolution stochastic optimization method. The proposed data model was tested and evaluated with the credit card dataset that contained transactions effected in September 2013 by the European cardholders. The dataset was downloaded from the Kaggle competition data repository (Kaggle, 2018).

The weighted voting stacking ensemble method was designed and used for the credit card dataset since the base models (machine learning methods) are different and performs differently. Finding the relevant weights for the base models has proven to be a challenging task. However, the differential evolution

method was employed to search the optimum weights for the weighted voting stacking ensemble method.

The differential evolution method was able to find the optimum weights for the weighted voting stacking ensemble method. When used in combination with differential evolution stochastic optimization method, the weighted voting stacking ensemble method performed far much better than the base models (machine learning methods) and the weighted voting stacking ensemble method. The true negative (TN) and the true positive (TP) transactions of the confusion matrix were high while the false negative (FN) and the false positive (FP) transactions were low.

This means that the weighted voting stacking ensemble method with differential evolution method was able to generalize the data distribution by classifying fraudulent and non-fraudulent transactions correctly, and the model is not biased. It can therefore be concluded that the weighted voting stacking ensemble method in combination with differential evolution stochastic optimization method was able to classify illegal and legal transactions correctly and efficiently.

Some researchers have managed to create machine learning methods for addressing the detection of credit card fraud (Malini & Pushpa 2017, pp. 1 – 4; Ganji 2012, pp. 1 – 5; Das, et al. 2017, pp. 1 – 4; Manjaramkar & Kokare 2017, pp. 1 – 4; Liu, et al. 2017, pp. 1 – 6; Mao, et al. 2017, pp. 1 - 8). However, their approach involves the individual use of the machine learning methods and a comparison of the output accuracies. It is argued in this research study that one machine learning method cannot, regardless of its computational power, be used to solve problems in all the cases. This argument is supported by the No-Free-Lunch (NFL) theorem, which states that there is no machine learning method that can achieve the best performance for all measures in every given application (Macready & Wolpert 1996, pp. 67 - 82).

The argument presented is this study is based on the fact that credit card fraudsters continue to upskill themselves with skills that allow them to bypass the powerful machine learning algorithms used in banks, and fraudsters exploit the loopholes present in every algorithm to their own benefit. Therefore, the creation of a detection

method that overcomes the loopholes found in the individual machine learning methods is seen as the best way for reducing or eliminating malicious activities associated with credit card fraud.

The approach adopted in this research study for combating credit card fraud was different. Various machine learning algorithms that performs differently were chosen. Various machine learning methods were chosen with the aim to cover the search space of the problem. However, the method to guide the researchers in terms of the machine learning methods being various enough to cover the search space of the problem is unknown and was not designed in this study. Various machine learning models were chosen to weight vote the final output of each transaction of the dataset. The voting of the machine learning models was designed to solve the problem of weak classifier models given a credit card transaction.

Since various machine learning method produced different prediction probability for each transaction, the weights for each voting machine learning model were used. However, to get the optimum weights for the machine learning techniques, it was necessary that the differential evolution method was used. The study has demonstrated that the differential evolution method is a good optimizer of weights for the machine learning models to vote the final output class of the credit card transactions.

6.2. Conclusions

The Main Research Question, which was outlined in Chapter 1, was: To obtain optimum data analytics algorithms that performs data insights to reveal fraudulent data patterns in order to prevent future fraudulent transactions. Dataset available from Kaggle machine learning competitions data repository (Kaggle, 2018) was used for addressing the objectives of the research study. This publicly available Kaggle repository data contained credit card dataset transactions made during September 2013 by European cardholders and was consisted of a total of 284 807 transactions, of which 0.172% (492) were illegal.

The Safe-Level SMOTE method was used for oversampling the fraudulent transactions while the duplicates were removed and random sampling was performed to non-fraudulent transactions to balance the class distribution. The feature selection

methods such as univariate selection algorithm, recursive feature elimination algorithm, and feature importance algorithm were stacked and majority voting was performed to vote out the feature(s) that did not add value to the decision making of the base models.

The weaknesses of the widely used, popular, and powerful base models selected for the detection of credit card malicious activities was examined through the designing, testing, and evaluation processes. An evaluation of the base models using the confusion matrix revealed that the performance expectancies of the base models were different and each base model performed well in specific areas or behaviours of the dataset. Furthermore, the data observations were different in nature, and their differences creates further research opportunities in areas of the credit card data whereby the individual base models are unable to detect fraudulent behaviours.

Various machine learning methods that perform differently and target various areas of the credit card data observations cover almost the entire search space of credit card data observations. To encourage proper credit card fraud detection or management system, financial institutions must design an environment that enables the adoption of weighted stacking ensemble methods with relevant base model weights to address the problem of credit card fraud.

In this research study, it is proposed that the weighted stacking ensemble method with differential evolution stochastic optimization model is suitable for the issue of illegal activities associated with credit card fraud. The DE approach produced better results because it can be easily applied on a variety of real valued problems where the feature vector space is multi-dimensional and where noise exist in the dataset. The noise was corrected by correcting the crossover vectors with values which are less than the lower bound and values which are more than the upper bound of the search space. Another important point why DE produced better results the hyper-parameters (CR and F) do not require the same fine tuning unlike in many other Evolutionary Algorithms where the same hyper-parameter fine tuning is required. This helps vectors in DE to evolve and converge towards the optimum weights of the stacking ensemble method.

The proposed model, which was evaluated using the confusion matrix, was found to perform better than the base models in terms of accuracy. Furthermore, the model was found to cover almost the entire search space of the credit card transactions.

However, it is noteworthy that the proposed model has its own limitations. Specifically, the framework for guiding the study in terms of the selection for stacking ensemble method of the base models covering the entire search space of the credit card data observations was not addressed.

It is recommended that future works should include the implementation of the framework to guide the financial institutions in terms of selection for stacking ensemble method of the base models that covers the entire search space of the credit card data observations. The framework should be flexible enough to cater for all types of credit card datasets.

7. References

Ahluwalia, S. & Mahajan, P., 2011. *Implications of new privacy and data protection rules.* [Online]
Available at: http://www.internationallawoffice.com/Newsletters/Banking/India/Amarchand-Mangaldas-Suresh-A-Shroff-Co/Implications-of-new-privacy-and-data-protection-rules
[Accessed 19 August 2011].

Ali, S., Tirumala, S. S. & Sarrafzadeh, A., 2015. Ensemble learning methods for decision making: Status and future prospects. *2015 International Conference on Machine Learning and Cybernetics (ICMLC),* pp. 211 - 216.

Altayar, M. & Almoqren, N., 2016. The motivations for big data mining technologies adoption in saudi banks. *4th Saudi International Conference on Information Technology (Big Data Analysis) (KACSTIT),* pp. 1 - 8.

Anbarasi, M. & Dhivya, S., 2017. Fraud Detection using Oulier predictor in health insurance Data. *INTERNATIONAL CONFERENCE ON INFORMATION, COMMUNICATION & EMBEDDED SYSTEMS,* pp. 1 - 6.

Andel, M. et al., 2015. Sparse Omics-network Regularization to Increase Interpretability and Performance of Linear Classification Models. *IEEE International Conference on Bioinformatics and Biomedicine,* pp. 1 - 6.

Anderson, R., 2017. 'Petroleum Analytics Learning Machine' For Optimizing the Internet of Things Of Today's Digital Oil Field-To-Refinery Petroleum System. *IEEE International Conference on Big Data (BIGDATA),* pp. 1 - 4.

Andrew, N., 2017. *Machine Learning.* [Online]
Available at: https://www.coursera.org/learn/machine-learning

Andropov, S., Guirik, A., Budko, M. & Budko, M., 2017. Network Anomaly Detection 8sing Artificial Neural Networks. *20th Conference of Open Innovations Association,* pp. 1 - 6.

Arief, H., Saptawati, G. & Asnar, Y., 2016. Fraud detection based-on data mining on Indonesian E-Procurement System (SPSE). *International Conference on Data and Software Engineering (ICoDSE),* pp. 1 - 6.

Babu, B. & Rajeshwari, U., 2016. Real-time credit card fraud detection using Streaming Analytics. *2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT),* pp. 439 - 444.

Babu, S., Vasavi, S. & Nagarjuna, K., 2017. Framework for Predictive Analytics as a Service using ensemble model. *IEEE 7th International Advance Computing Conference,* pp. 121 - 128.

Ball, G. & Flamm, B., 1996. *1960's Sowing the seeds of the calculator revolution.* [Online] Available at: http://www.vintagecalculators.com/html/history_of_electronic_calculat.html

Bardwell, L. & Fearnhead, P., 2017. Bayesian detection of abnormal segments in multiple time series. *international society of bayesian analysis,* Volume 12, pp. 193-218.

Basha, S., Sahlol, A., Baz, S. & Hassanien, A., 2017. Neutrosophic Rule-based prediction system for assessment of pollution on benthic foraminifera in burullus Lagoon in egypt. *12th International Conference on Computer Engineering and Systems (ICCES),* pp. 1 - 6.

Basheer, I. & Hajmeer, M., 2000. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods,* pp. 1 - 29.

Behadada, O. et al., 2016. Logistic Regression Multinomial for Arrhythmia Detection. *IEEE 1st International Workshops on Foundations and Applications of Self-Systems,* pp. 1 - 5.

Bouteldja, M. A. & Batouche, M., 2017. A study on Differential Evolution and Cellular Differential Evolution for multilevel color image segmentation. *2017 Intelligent Systems and Computer Vision (ISCV),* pp. 1 - 8.

Brest, J. et al., 2006. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems.. *IEEE Transactions on Evolutionary Computation,* pp. 646 - 657.

Brownlee, J., 2016. *Supervised and Unsupervised Machine Learning Algorithms.* [Online] Available at: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Bunkhumpornpat, C., Sinapiromsaran, K. & Lursinsap, C., 2009. Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-Sampling TEchnique for Handling the Class Imbalanced Problem. *Pacific-Asia Conference on Knowledge Discovery and Data Mining,* pp. 475 - 482.

Bunkhumpornpat, C., Sinapiromsaran, K. & Lursinsap, C., 2011. MUTE: Majority under-sampling technique. *2011 8th International Conference on Information, Communications & Signal Processing,* pp. 1 - 4.

Bunkhumpornpat, C. & Subpaiboonkit, S., 2013. Safe level graph for synthetic minority over-sampling techniques. *2013 13th International Symposium on Communications and Information Technologies (ISCIT),* pp. 570 - 575.

Calma, A. et al., 2016. From Active Learning to Dedicated Collaborative Interactive Learning. *International Conference on Architecture of Computing Systems,* pp. 1 - 8.

Cao, M. & Guo, C., 2017. Research on the Improvement of Association Rule Algorithm for Power Monitoring Data Mining. *10th International Symposium on Computational Intelligence and Design,* pp. 1-4.

Chaaya, G. & Maalouf, H., 2017. Anomaly Detection on a Real-Time Server Using Decision Trees. *8th International Conference on Information Technology,* pp. 1 - 7.

Chase, M., 2018. *The state of data management in the public sector in 2018.* [Online] Available at: https://www.edq.com/blog/the-state-of-data-management-in-the-public-sector-in-2018/

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P., 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research,* pp. 321 - 357.

Cheng, C., Xu, W. & Wang, J., 2012. A Comparison of Ensemble Methods in Financial Market Prediction. *2012 Fifth International Joint Conference on Computational Sciences and Optimization,* pp. 755 - 759.

Chezian, R. & Devi., C., 2016. A Relative Evaluation of the Performance of Ensemble Learning in Credit Scoring. *IEEE International Conference on Advances in Computer Applications (ICACA),* pp. 161 - 165.

Cui, B. et al., 2018. Superpixel-Based Extended Random Walker for Hyperspectral Image Classification. *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING,* pp. 1 - 11.

Dalvi, P. T. & Vernekar, N., 2016. Anemia detection using ensemble learning techniques and statistical models. *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT),* pp. 1747 - 1751.

Das, T., Tripathy, A. & Mishra, A., 2017. Optical Character Recognition using Artificial Neural Network. *International Conference on Computer Communication and Informatic,* pp. 1 - 4.

Dehghan, A. et al., 2017. Predicting Likelihood of Requirement Implementation within the Planned Iteration: An Empirical Study at IBM. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR),* pp. 124 - 134.

Ding, L., Sun, Y. & Xiong, Z., 2017. Early Chatter Detection based on Logistic Regression with Time and Frequency Domain Features. *IEEE International Conference on Advanced Intelligent Mechatronics,* pp. 1 - 6.

Domingo, B., Ponnambalam, S. & Kanagaraj, G., 2013. A Differential Evolution Based Algorithm for Single Container Loading Problem. *2013 IEEE Symposium on Differential Evolution (SDE),* pp. 105 - 111.

Eurofinas & ACCIS, 2011. *Fraud Prevention and Data Protection,* s.l.: Eurofinas.

Flood, D., Jagadish, H. & Raschid, L., 2016. Big data challenges and opportunities in financial stability monitoring. *Banque de France, Financial Stability Review,* pp. pp. 1-20.

Funaki, R. & Takagi, H., 2011. Application of Gravity Vectors and Moving Vectors for the Acceleration of both Differential Evolution and Interactive Differential Evolution. *2011 Fifth International Conference on Genetic and Evolutionary Computing,* pp. 287 - 290.

Ganji, V., 2012. Credit card fraud detection using anti-k nearest neighbor algorithm. *International Journal on Computer Science and Engineering,* pp. 1 - 5.

Gestel, T. et al., 2011. Financial Time Series Prediction Using Least Squares Support Vector Machines Within the Evidence Framework. *IEEE TRANSACTIONS ON NEURAL NETWORKS,* pp. 1 - 13.

Giannakis, G., 2018. Sketched Subspace Clustering. *IEEE TRANSACTIONS ON SIGNAL PROCESSING,* Volume 66, pp. 1- 13.

Godase, A. & Attar, V., 2012. Classification of data streams with skewed distribution. *IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS),* pp. 1 - 6.

Gomes, F. et al., 2016. Heuristic Regression Method for Descriptive Data Analysis. *IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC),* pp. 1-5.

Gosain, A. & Sardana, S., 2017. Handling class imbalance problem using oversampling techniques: A review. *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI),* pp. 79 - 85.

Gosain, A. & Sardana, S., 2017. Handling Class Imbalance Problem using Oversampling Techniques: A Review. *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI),* pp. 79 - 85.

Goudos, S. K. et al., 2016. Optimization of Power Consumption in Wireless Access Networks Using Differential Evolution with Eigenvector Based Crossover Operator. *2016 10th European Conference on Antennas and Propagation (EuCAP),* pp. 1 - 4.

Government, G., 2013. Act No.4 of 2013: Protection of Personal Information Act. 26 Novermber, pp. pp.1-149.

Grace, E., Rai, A. & Redmiles, E., 2016. Detecting Fraud, Corruption, and Collusion in International Development Contracts: The Design of a Proof-of-Concept Automated System. *IEEE International Conference on Big Data (Big Data),* pp. pp.1-10.

Gunasekaran, A. et al., 2017. Big data and predictive analytics for supply chain and organizational performance. *Journal of Business Research,* Volume 70, pp. 308-317.

Guo, X., Yin, Y., Zhou, G. & Dong, C., 2008. Solving Cross-selling Problems with Ensemble Learning: A Case Study. *International Conference on Advanced Computer Theory and Engineering,* pp. pp.1-6.

Guoying, Z. & Wenge, L., 2010. Evaluation of China's Oil Security Based on Support Vector Machine（SVM Theory. *International Conference of Information Science and Management Engineerin,* pp. 1 - 4.

Haneem, F., Ali, R., Kama, N. & Basri, S., 2017. Descriptive Analysis and Text Analysis in Systematic Literature Review: A review of Master Data Management. *International Conference on Research and Innovation in Information Systems (ICRIIS),* pp. 1-6.

Han, X., Xu, L., Ren, M. & Gu, W., 2015. A Naive Bayesian network intrusion detection algorithm based on Principal Component Analysis. *7th International Conference on Information Technology in Medicine and Education,* pp. 1 - 4.

Hoffman, C., 2017. *How Credit Card Skimmers Work, and How to Spot Them.* [Online] Available at: https://www.howtogeek.com/193958/atm-skimmers-explained-how-to-protect-your-atm-card/

Hsu, C.-M. & Chao, H.-M., 2007. An Online Fraud-Resistant Technology for Credit Card E-Transactions. *IEEE Region 10 Conference,* pp. 1 - 4.

Hui, S. & Suganthan, P. N., 2013. Ensemble Crowding Differential Evolution with Neighborhood Mutation for Multimodal Optimization. *2013 IEEE Symposium on Differential Evolution (SDE),* pp. 135 - 142.

Infromatica, 2017. *Informatica.* [Online]
Available at: https://www.informatica.com/services-and-training/glossary-of-terms/data-preparation-definition.html#fbid=LbxGDp-7XKa

Jensen, N. et al., 2015. A ScalablevPrescriptive Parallel Debugging Model. *IEEE 29th International Parallel and Distributed Processing Symposium,* pp. 1 - 11.

Jia, O. et al., 2017. Joint Sparse Auto-encoder: A Semi-supervised Spatio-temporal Approach in Mapping Large-scale Croplands. *IEEE International Conference on Big Data,* pp. 1 - 10.

John, S. et al., 2016. Realtime fraud detection in the banking sector using data mining techniques/algorithm. *International Conference on Computational Science and Computational Intelligence,* pp. 1186 - 1191.

kaggle, 2018. *Kaggle dataset.* [Online]
Available at: https://www.kaggle.com/datasets

Kalajac, E., Karabegović, A. & Ponjavić, M., 2018. Optimization of the structures locations using a genetic algorithm in the transmission line design. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO),* pp. 883 - 886.

Kamal, A., Zhu, X., Pandya, A. & Hsu, S., 2009. Feature Selection with Biased Sample Distributions. *IEEE International Conference on Information Reuse & Integration,* pp. 1 - 6.

kareem, S., Ahmad, R. & Sarlan, A., 2017. Framework for the Identification of Fraudulent Health Insurance Claims using Association Rule Mining. *IEEE Conference on Big Data and Analytic,* pp. 1 - 6.

Katkar, V. & Kulkarni, S., 2013. Experiments on Detection of Denial of Service Attacks using Naive Bayesian Classifier. *International Conference on Green Computing, Communication and Conservation of Energy (ICGCE),* pp. 1 - 6.

Keith, F., 2017. *A Brief History of Big Data.* [Online]
Available at: http://www.dataversity.net/brief-history-big-data/

Kho, J. & Vea, L., 2017. Credit card fraud detection based on transaction behavior. *IEEE Region 10 Conference,* pp. 1 - 5.

Khoshgoftaar, T., Allen, E. & Deng, J., 2001. Controlling Overfitting in Software Quality Models: Experiments with Regression Trees and Classification. *Proceedings Seventh International Software Metrics Symposium,* pp. 1 - 9.

Legal_Information_Institute, n.d. *Credit Card Fraud.* [Online]
Available at: https://www.law.cornell.edu/wex/credit_card_fraud

Li, M., Liu, W. & Li, H., 2018. Probabilistic Regularized Extreme Learning Machine for Robust Modeling of Noise Data. *IEEE TRANSACTIONS ON CYBERNETICS,* pp. 1 - 10.

Li, R. & Wang, X., 2017. Self-adaptive weighted majority vote algorithm based on entropy. *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS),* pp. 73 - 77.

Lita, I., Visan, D., Oprea, S. & Cioc, B., 2007. Hardware Design for Noise Reduction in Data Acquisition Modules. *30th International Spring Seminar on Electronics Technology,* pp. 1 - 5.

Liu, H. & Dash, M., 1997. Feature Selection for Classification. *Intelligent Data Analysis,* 1(3), pp. 131-156.

Liu, R. et al., 2018. An Ensemble of Classifiers Based on Positive and Unlabeled Data in One-Class Remote Sensing Classification. *IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING,* pp. 1 - 13.

Liu, X., Wei, W. & Yu, F., 2007. SVM Theory and Its Application in Fault Diagnosis of HVDC System. *Third International Conference on Natural Computation, 2007. ICNC 2007,* pp. 1 - 5.

Liu, Y. & Khoshgoftaar, T., 2004. Reducing overfitting in genetic programming models for software quality classification. *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.,* pp. 1 - 10.

Liu, Y. et al., 2017. On Over-Exposed Region Detection with Regularized Logistic Regression. *10th International Conference on Ubi-media Computing and Workshop,* pp. 1 - 6.

Li, X., Chang, W., Zhou, S. & Wei, F., 2017. The Panel Data Predictive Model for Recurrence of Cerebral Infarction with Health Care Data Analysis. *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM),* pp. 1 - 6.

Ll, L., Huang, D. & Wang, M., 2011. Protein-Protein Interaction Extraction Based on Ensemble Kernel Model and Active Learning Strategy. *7th International Conference on Natural Language Processing and Knowledge Engineering,* pp. 1 - 6.

Macready, W. & Wolpert, D., 1996. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation,* pp. 67 - 82.

Madathil, D., Pandi, V. R., Ilango, K. & Nair, M. G., 2017. Differential evolution based energy management system for zero net energy building. *2017 International Conference on Technological Advancements in Power and Energy (TAP Energy),* pp. 1 - 5.

Mak, L., Ng, G., Lim, G. & Mao, K., 2011. A merging Fuzzy ART clustering algorithm for overlapping data. *IEEE Symposium on Foundations of Computational Intelligence (FOCI),* pp. 1 - 6.

Malini, N. & Pushpa, M., 2017. Analysis on Credit Card Fraud Identification Techniques based on KNN and Outlier Detection. *3rd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics,* pp. 1 - 4.

Manjaramkar, A. & Kokare, M., 2017. Decision Trees for Microaneurysms Detection In Color Fundus Images. *IEEE International Conference on Innovations in Green Energy and Healthcare Technologies,* pp. 1 - 4.

Mao, X., Wang, Y., Liu, X. & Guo, Y., 2017. A Hybrid Feedforward-feedback Hysteresis Compensator in Piezoelectric Actuators Based on Least Squares Support Vector Machine. *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS,* pp. 1 - 8.

Matsuda, K. & Murase, K., 2016. Single-Layered Complex-Valued Neural Network with SMOTE for Imbalanced Data Classification. *2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS),* pp. 349 - 354.

Meidianingsih, Q., Erfiani & Sartono, B., 2017. Study of Safe-Level SMOTE Method in Unbalanced Data Classification Cases. *International Journal of Scientific & Engineering Research Volume 8, Issue 5, May-2017 ,* pp. 1167 - 1171.

Meiping, X., 2009. Application of Bayesian Rules Based on Improved K-Means Cassification on Credit Card. *International Conference on Web Information Systems and Mining,* pp. 1 - 4.

Mei, X. & Jiang, Y., 2016. Association Rule-based Feature Selection for Credit Risk Assessment. *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS),* pp. 301 - 305.

Mei, X. & Jiang, Y., 2016. Association Rule-based Feature Selection for Credit Risk Assessment. *IEEE International Conference of Online Analysis and Computing Science (ICOACS),* pp. 301 - 305.

Miholca, D. & Onicaş, A., 2017. Detecting depression from fMRI using Relational Association Rules and Artificial Neural Networks. *13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP),* pp. 1 - 8.

Mitik, M. et al., 2016. Data Mining based Product Marketing Technique for Banking Products. *IEEE 16th International Conference on Data Mining Workshops,* pp. 552 - 559.

Mitik, M. et al., 2016. Data Mining based Product Marketing Technique for Banking Products. *2016 IEEE 16th International Conference on Data Mining Workshops,* pp. 552 - 559.

Monk, A., 2016. Practical Applications of How Technology Will Change the Business of Investing. *69th CFA Institute Annual Conference,* pp. pp. 1-3.

Munar, A., Chiner, E. & Sales, I., 2014. A Big Data Financial Information Management Architecture for Global Banking. *International Conference on Future Internet of Things and Cloud,* pp. 1-4.

Mu, X., Watta, P. & Hassoun, M., 2005. Combining local similarity measures: summing, voting, and weighted voting. *2005 IEEE International Conference on Systems, Man and Cybernetics,* Volume 3, pp. 2661 - 2666.

Nizar, A. H., Dong, Z. Y. & Zhang, P., 2008. Detection Rules for Non Technical Losses Analysis in Power Utilities. *2008 IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century,* pp. 1 - 8.

Nyikes, Z. & Rajnai, Z., 2015. Big Data, As Part of the Critical Infrastructure. *IEEE 13th International Symposium on Intelligent Systems and Informatic,* pp. 217 - 222.

Pang, S., He, Y. & Wang, L., 2016. The Intelligent Control Model and Application for Commercial Bank Systems Emergence under Risk Status Based on Big Data. *12th International Conference on Computational Intelligence and Security,* pp. 1-5.

Patil, A. & Gupta, P., 2017. A Review on Up-growth algorithm using Association Rule mining. *International Conference on Computing Methodologies and Communication,* pp. 1-4.

Patwardhan, R., 2016. *Data Analytics: Which one is correct for me – Descriptive, Predictive, or Prescriptive?.* [Online]
Available at: http://www.rtap.io/blog/data-analytics-which-one-is-correct-for-me-descriptive-predictive-or-prescriptive/
[Accessed 09 06 2016].

Pavlyshenko, B., 2016. Machine Learning, Linear and Bayesian Models for Logistic Regression in Failure Detection Problems. *IEEE International Conference on Big Data,* pp. 1 - 5.

Pengfei, J., Chunkai, Z. & Zhenyu, H., 2014. A new sampling approach for classification of imbalanced data sets with high density. *2014 International Conference on Big Data and Smart Computing (BIGCOMP),* pp. 217 - 222.

Peng, Y. et al., 2017. Negative ε Dragging Technique for Pattern Classification. *IEEE Access,* pp. 1 - 7.

Pushpa, M. & Malini, N., 2017. Analysis on Credit Card Fraud Identification Techniques based on KNN and Outlier Detection. *3rd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEEICB17),* pp. 255 - 258.

Pushpa, M. & Malini, N., 2017. Analysis on Credit Card Fraud Identification Techniques based on KNN and Outlier Detection. *3rd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEEICB17),* pp. 255 - 258.

Queiroz, J., Leitão, P. & Dias, A., 2016. Predictive data analysis driven multi-agent system approach for electrical micro grids management. *IEEE 25th International Symposium on Industrial Electronics (ISIE),* pp. 1 - 6.

Rajaei, R., Shafai, B. & Ramezani, A., 2017. A top-down scheme of descriptive time series data analysis for healthy life. *IEEE High Performance Extreme Computing Conference (HPEC),* pp. 1 - 6.

Rajak, I. & Mathai, K., 2015. Intelligent Fraudulent Detection System Based SVM And Optimized By Danger Theory. *IEEE International Conference on Computer, Communication and Control,* pp. 1 - 4.

Reddy, R., Ramadevi, Y. & Sunitha, K., 2017. Enhanced Anomaly Detection using ensemble support vector machine.. *InternationalConference On Big Data Analytics and computationalIntelligence,* pp. 1 - 5.

Rijmenam, M., 2017. *The Future of Big Data? Three Use Cases of Prescriptive Analytics.* [Online]
Available at: https://www.linkedin.com/pulse/future-big-data-three-use-cases-prescriptive-mark-van-rijmenam

Rout, S., Mishra, S. & Mishra, S., 2017. A review on application of artificial neural network(ANN) on protein secondary structure prediction. *Second International Conference on Electrical, Computer and Communication Technologies (ICECCT),* pp. 1 - 6.

Rushin, G. et al., 2017. Horse Race Analysis in Credit Card Fraud—Deep Learning, Logistic Regression, and Gradient Boosted Tree. *Systems and Information Engineering Design Symposium (SIEDS),* pp. 117 - 121.

Salazar, A., Safont, G. & Vergara, L., 2014. Surrogate techniques for testing fraud detection algorithms in credit card operations. *International Carnahan Conference on Security Technology (ICCST),* pp. 1 - 6.

Silva, L. & Wunsch, D., 2017. Validity index-based vigilance test in adaptive resonance theory neural networks. *IEEE Symposium Series on Computational Intelligence (SSCI),* pp. 1 - 8.

Singh, P., Jain, N. & Maini, A., 2015. Investigating the Effect Of Feature Selection and Dimensionality Reduction On Phishing Website Classification Problem. *IEEE International Conference on Advances in Computer Applications (ICACA),* pp. 388 - 393.

Singh, P., Jain, N. & Maini, A., 2015. Investigating the Effect Of Feature Selection and Dimensionality Reduction On Phishing Website Classification Problem. *2015 1st International Conference on Next Generation Computing Technologies (NGCT),* pp. 388 - 393.

Smolyakov, V., 2017. *Ensemble Learning to Improve Machine Learning Results.* [Online] Available at: https://blog.statsbot.co/ensemble-learning-d1dcd548e936

Somasundaram, A. & Reddy, U., 2017. Modelling a Stable Classifier for Handling Large Scale Data with Noise and Imbalance. *International Conference on Computational Intelligence in Data Science,* pp. 1 - 6.

Srinivas, V. S., Srikrishna, A. & Reddy, B. E., 2018. Automatic Feature Subset Selection for Clustering Images using Differential Evolution. *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR),* pp. 216 - 217.

Srivastava, A. et al., 2016. Credit Card Fraud Detection at Merchant Side using Neural Networks. *3rd International Conference on Computing for Sustainable Global Development (INDIACom),* pp. 1 - 4.

Staff, W., 2017. *The biggest types of credit card fraud in South Africa.* [Online] Available at: https://businesstech.co.za/news/banking/172249/the-biggest-types-of-credit-card-fraud-in-south-africa/

TCI, 2015. *Driving profitable growth through improved data insights and segmentation practices for the future customer and digital bank.* Indaba Hotel▪Sandton, Trade Conferences International, pp. pp. 1-7.

Todd, D. & Harvey, M., 2015. Automated Feature Design for Numeric Sequence Classification by Genetic Programming. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION,* 19(4), pp. 474 - 489.

Toporek, D., Power, G. & Hutchings, A., 2011. Making Fault Data and Non-SCADA Data Accessible for Predictive Analysis in Data Historians. *IEEE Power and Energy Society General Meeting,* pp. 1 - 4.

Trelewicz, J., 2017. Big Data and Big Money - The Role of Data in the Financial Sector. *IEEE Computer Society,* pp. 1 - 3.

Verma, A. & Mehta, S., 2017. A comparative study of ensemble learning methods for classification in bioinformatics. *2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence,* pp. 155 - 158.

Vipani, R., Hore, S., Basak, S. & Dutta, S., 2017. Detection of Epilepsy using Hilbert Transform and KNN Based Classifier. *IEEE International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials,* pp. 1 - 5.

Wang, G. et al., 2010. Predicting credit card holder churn in banks of China using data mining and MCDM. *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology,* pp. 215 - 218.

Wang, J. et al., 2017. From Partition-Based Clustering to Density-Based Clustering: Fast Find Clusters With Diverse Shapes and Densities in Spatial Databases. *IEEE Access,* pp. 1 - 12.

Wang, T., Fang, C. V. & Chun-Ming Lai, S. W., 2015. Triaging Anomalies in Dynamic Graphs: Towards Reducing False Positives. *2015 IEEE International Conference on Smart City/SocialCom/SustainCom together with DataCom 2015 and SC2 2015,* pp. 354 - 359.

Wang, X., NiBai, Y., Zhang, R. & Wei, X., 2017. The Research of Entertainers Collaboration Relation Based On Association Rules. *10th International Symposium on Computational Intelligence and Desig,* pp. 1-4.

Wang, Y. & Hajli, N., 2017. Exploring the path to big data analytics success in healthcare. *Journal of Business Research,* Volume 70, pp. 287-299.

West, J. & Bhattacharya, M., 2016. An Investigation on Experimental Issues in Financial Fraud Mining. *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA),* pp. 1796 - 1801.

West, J. & Bhattacharya, M., 2016. Some Experimental Issues in Financial Fraud Mining. *The International Conference on Computational Science (ICCS),* p. 1734 – 1744.

Woodford, C., 2017. *A brief history of computers.* [Online]
Available at: http://www.explainthatstuff.com/historyofcomputers.html

Woten, D. & Shenawee, M., 2007. Improvement of artificial neural network detection of breast cancer using broadband dual polarized antenna. *IEEE Antennas and Propagation Society International Symposium,* pp. 1 - 4.

Writer_Staff, 2017. *The biggest types of credit card fraud in South Africa.* [Online]
Available at: https://businesstech.co.za/news/banking/172249/the-biggest-types-of-credit-card-fraud-in-south-africa/

Writer, S., 2015. *Watch out for these card skimming and PIN theft tricks criminals use.* [Online]
Available at: https://mybroadband.co.za/news/banking/128194-watch-out-for-these-card-skimming-and-pin-theft-tricks-criminals-use.html

Wu, P. & Yang, C., 2017. The Green Fleet Optimization Model for A Low-Carbon Economy: APrescriptive Analytics. *Proceedings of the 2017 IEEE International Conference on Applied System Innovation,* pp. 1 - 4.

Xu, L., Luo, B., Yu, F. & Xie, J., 2011. A novel description of the reproducing kernel support vector machines. *IEEE International Conference on Computer Science and Automation Engineering,* pp. 1 - 5.

Yuan, H., Ding, B. & Sun, Z., 2008. Workflow Exception Forecasting Method Based on SVM Theory. *International Symposium on Computational Intelligence and Design,* pp. 1 - 6.

Yu, Y. et al., 2017. A combinatorial clustering method for sequential fraud detection. *International Conference on Service Systems and Service Management,* pp. 1 - 6.

Zakerian, A., Maleki, A., Mohammadnian, Y. & Amraee, T., 2017. Bad data detection in state estimation using Decision Tree technique. *25th Iranian Conference on Electrical Engineering,* pp. 1 - 6.

Zaza, S. & Al-Emran, M., 2015. Mining and Exploration of Credit Cards Data in UAE. *Fifth International Conference on e-Learning,* pp. 275 - 279.

Zeager, M. et al., 2017. Adversarial Learning in Credit Card Fraud Detection. *Systems and Information Engineering Design Symposium (SIEDS),* pp. 112 - 116.

Zhang, J., Chen, W., Gao, M. & Shen, G., 2017. Mitigating Fiber Nonlinearity Using Support Vector Machine with Genetic Algorithm. *Conference on Lasers and Electro-Optics Pacific Rim (CLEO-PR),* pp. 1 - 3.

Zhdanova, M. et al., 2014. No Smurfs: Revealing Fraud Chains in Mobile Money Transfers. *9th International Conference on Availability, Reliability and Security,* pp. 11 - 20.

Zhou, P. et al., 2014. A Two-Stage Classification Framework for Imbalanced Data with Overlapping Labels. *Proceedings of 2014 IEEE International Conference on Service Operations and Logistics, and Informatics,* pp. 1 - 6.

Zirpe, S. & Joglekar, B., 2017. Negation Handling using Stacking Ensemble Method. *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA),* pp. 1 - 5.

Chu, L. et al., 2019. *Feature Importance Identification through Bottleneck Reconstruction.* Milan, Italy, Italy, IEEE.

Curtis, J. & Kon, M., 2015. *Class Discovery via Bimodal Feature Selection in Unsupervised Settings.* Miami, FL, USA, IEEE.

Drotár, P. & Gazda, J., 2016. *Two-Step Feature Selection Methods for Selection of Very Few Features.* IEEE, Dubai, United Arab Emirates, pp. 1-5.

Dutta, D., Paul, D. & Ghosh, P., 2018. *Analysing Feature Importances for Diabetes Prediction using Machine Learning.* Vancouver, BC, Canada, IEEE.

Lv, X., Wu, J. & Liu, W., 2014. *Face Image Feature Selection Based on Gabor Feature and Recursive Feature Elimination.* Hangzhou, China, IEEE.

Qiu, C. et al., 2018. *Feature Importance Analysis of Sentinel-2 Imagery for Large-Scale Urban Local Climate Zone Classification.* Valencia, Spain, IEEE.

Subho, M. R. H. et al., 2019. *A Univariate Feature Selection Approach for Finding Key Factors of Restaurant Business.* Kolkata, India, India, IEEE.

Tran, H. D. & Li, H., 2009. *Sound event classification based on Feature Integration, Recursive Feature Elimination and Structured Classification.* Taipei, Taiwan, IEEE.

Zeng, X., Chen, Y.-W., Tao, C. & Alphen, D. v., 2009. *Feature Selection Using Recursive Feature Elimination for Handwritten Digit Recognition.* Kyoto, Japan, IEEE.

8. Appendix

8.1. Ethical Clearance

UNISA

UNISA COLLEGE OF SCIENCE, ENGINEERING AND TECHNOLOGY'S (CSET) RESEARCH AND ETHICS COMMITTEE

01 November 2019

Ref #: 067/KMD/2019/CSET_SOC
Name: Mr Kgaugelo Moses Dolo
Student #: 58527141

Dear Mr Kgaugelo Moses Dolo

Decision: Ethics Approval for 3 years
(No Humans involved)

Researchers: Mr Kgaugelo Moses Dolo, 58527141@mylife.unisa.ac.za, +27 72 382 6913; +27 71 342 3198

Project Leader(s): Prof Ernest Mnkandla, mnkane@unisa.ac.za, +27 11 670 9059

Working title of Research:
Differential Evolution technique on weighted voting stacking ensemble method for credit card fraud detection

Qualification: MSc in Computing

Thank you for the application for research ethics clearance by the Unisa College of Science, Engineering and Technology's (CSET) Research and Ethics Committee for the above-mentioned research. Ethics approval is granted for a period of three years, from 01 November 2019 to 01 November 2022.

1. The researcher will ensure that the research project adheres to the values and principles expressed in the UNISA Policy on Research Ethics.
2. Any adverse circumstance arising in the undertaking of the research project that is relevant to the ethicality of the study, as well as changes in the methodology, should be communicated in writing to the Unisa College of Science, Engineering and Technology's (CSET) Research and Ethics Committee. An amended application could

RECEIVED
2019 -11- 13
Office of the Deputy Executive Dean
College of Science, Engineering & Technology

RECEIVED
2019 -11- 14
OFFICE OF THE EXECUTIVE DEAN
College of Science, Engineering and Technology

Open Rubric

be requested if there are substantial changes from the existing proposal, especially if those changes affect any of the study-related risks for the research participants.

3. The researcher(s) will conduct the study according to the methods and procedures set out in the approved application.
4. Any changes that can affect the study-related risks for the research participants, particularly in terms of assurances made with regards to the protection of participants' privacy and the confidentiality of the data, should be reported to the Committee in writing, accompanied by a progress report.
5. The researcher will ensure that the research project adheres to any applicable national legislation, professional codes of conduct, institutional guidelines and scientific standards relevant to the specific field of study. Adherence to the following South African legislation is important, if applicable: Protection of Personal Information Act, no 4 of 2013; Children's act no 38 of 2005 and the National Health Act, no 61 of 2003.
6. Only de-identified research data may be used for secondary research purposes in future on condition that the research objectives are similar to those of the original research. Secondary use of identifiable human research data requires additional ethics clearance.
7. Submission of a completed research ethics progress report will constitute an application for renewal of Ethics Research Committee approval.

Note:

The reference number 067/KMD/2019/CSET_SOC should be clearly indicated on all forms of communication with the intended research participants, as well as with the Unisa College of Science, Engineering and Technology's (CSET) Research and Ethics Committee.

Yours sincerely

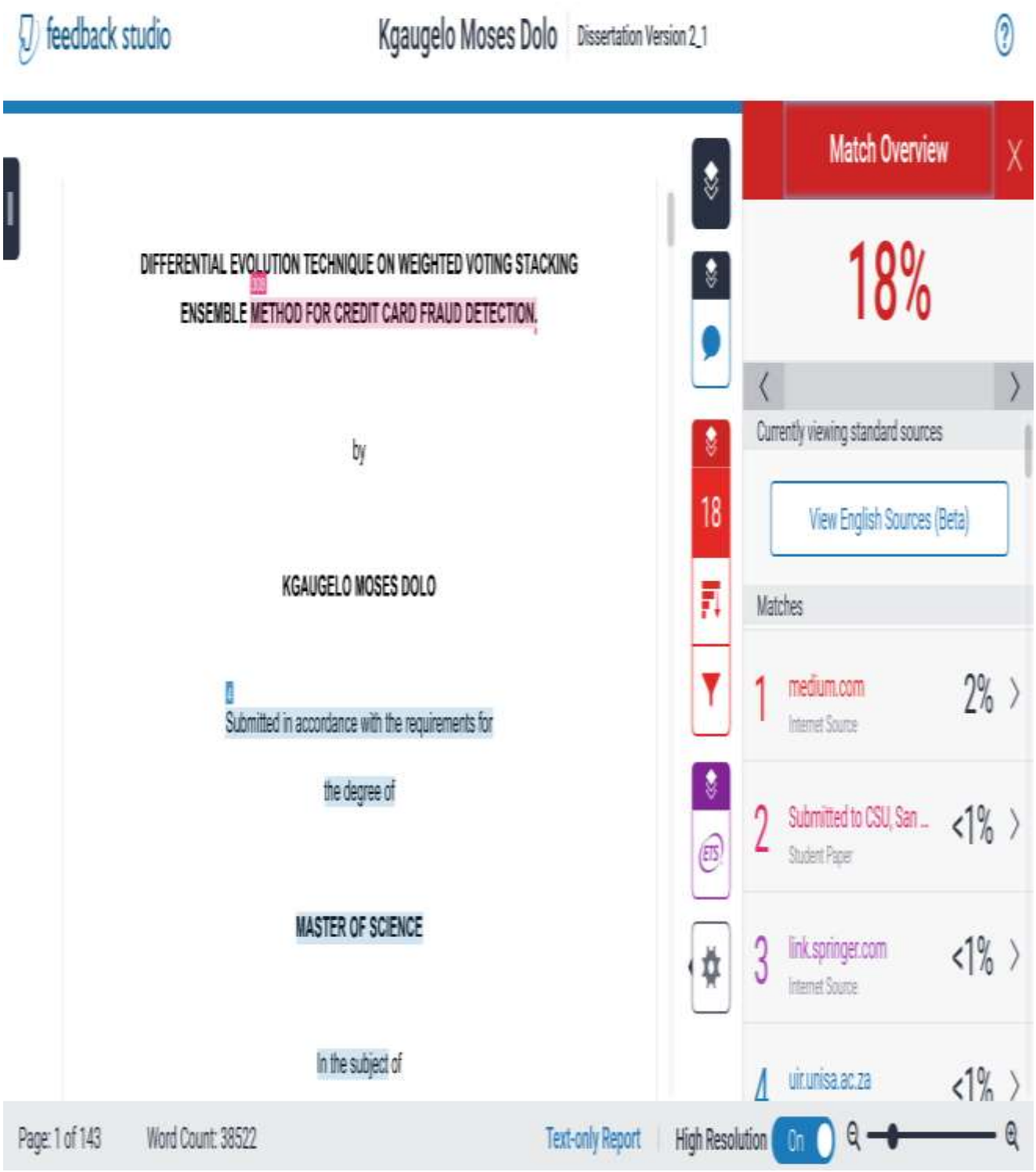
Dr. B Chimbo
Chair: Ethics Sub-Committee SoC, College of Science, Engineering and Technology (CSET)

Dr. MG Katumba
Director: School of Computing, CSET

Prof. B Mamba
Executive Dean: CSET

University of South Africa
Preller Street, Muckleneuk Ridge, City of Tshwane
PO Box 392 UNISA 0003 South Africa
Telephone: +27 12 429 3111 Facsimile: +27 12 429 4150
www.unisa.ac.za

8.2. Turnitin Report



8.3. Editorial Certificate

THE EDITING DOCTOR (PTY) LTD

COMPANY REGISTRATION NO.: 2017/053073/07; ADDRESS: 48 Acacia Glen, Trehant Road, Ormonde, 2091, Republic of South Africa;
MOBILE: + 27 (0) 61 544 2641; WHATSAPP: + 27 (0) 61 540 6472; E-MAIL: info@TheEditingDoctor.com

EDITORIAL CERTIFICATE

This certificate serves to confirm that the manuscript listed below was edited by one or more professional scientific editors with a PhD. The University or the student's supervisor(s) can contact us for a copy of the edited document that was submitted to the author(s).

Manuscript Title:

Differential evolution technique on weighted voting stacking ensemble method for credit card fraud detection

Authors:

Kgaugelo Dolo

Date Issued:

23 December 2019

Certificate Number:

20-112019PVT

TheEditingDoctor (Pty) Ltd works with researchers from academic and research organizations in the fields of science, engineering and technology to get their work ready for examination and/or publication in internationally acclaimed English journals. Our areas of specialization include Editing (i.e. editing poor syntax, poor general structure, repetition, redundancy, contradiction or inconsistency) and Rewriting (i.e. rewriting of phrases, sentences, paragraphs or sections to achieve better flow, more coherence clearer logic, and more appropriate vocabulary).

8.4. Python Source Codes

Algorithm 1: Remove Duplicates in the Training Dataset

```
def _duplicates(Training_data0):
    Training_data0_df = pd.DataFrame(Training_data0)
    Training_data0_df.drop_duplicates(keep='first', inplace=True)
    Num_removed = len(Training_data0) - len(Training_data0_df)
    return Num_removed, Training_data0_df
```

Algorithm 2: Random Sampling

```
def _randomSamples(Training_data0_df):
    reduced = []
    index   = []
    for i in range(700):
        ind = np.random.randint(0, len(training_data0_df))
        if ind not in index:
            index.append(ind)
            reduced.append(training_data0_df.iloc[ind])
        else:
            i -= 1
    return reduced
```

Algorithm 3: Combine Training Dataset without Duplicates with Random Sample Dataset

```
def _recombination(training_data , Training_data1):
    randomSamples = pd.DataFrame(_randomSamples(training_data))
    recombination = randomSamples.append(Training_data1)
    return recombination
```

Algorithm 4: Safe Level SMOTE

```
def euclidean_distance (x, y):
    return np.sqrt(sum( pow( a-b, 2 ) for a, b in zip( x , y )))

def _safe_level_smote(recombination):
    New_data = []
    for Pos_instance in [j for j,i in enumerate(np.array(recombination)) if
float(i[-1]) == 1.0]:
        k_distances                                                    =
[[euclidean_distance(np.array(recombination)[Pos_instance],
np.array(recombination)[i]), i] for i in range(len(np.array(recombination)))
if i != Pos_instance]
        k_distances.sort(reverse = False)
        k  =  [np.array(recombination)[j]  for  j  in  [i[1]  for  i  in
k_distances[0:5]]]
        n = k.pop(np.random.randint(0,5))
        n_distances = [[euclidean_distance(n, np.array(recombination)[i]),
i] for i in range(len(np.array(recombination)))]
        n_distances.sort(reverse = False)
        n_neighbors = [np.array(recombination)[j] for j in [i[1] for i in
n_distances[1:6]]]
        SLp = len([i for i in k if i[-1] == 1])
```

```
        SLn = len([i for i in n_neighbors if i[-1] == 1])
        if SLn != 0:
            SL_ratio = SLp/SLn
        else:
            SL_ratio = np.inf
        if SL_ratio == np.inf and SLp == 0:
            continue
        else:
            feature  = []
            for i in range(len(np.array(recombination)[0])):
                if SL_ratio == np.inf and SLp != 0:
                    gap = 0
                elif SL_ratio == 1:
                    gap = np.random.uniform(0, 1)
                elif SL_ratio > 1:
                    gap = np.random.uniform(0, 1/SL_ratio)
                elif SL_ratio < 1:
                    gap = np.random.uniform(1 - SL_ratio, 1)
                dif = k[0][i] - n_neighbors[0][i]
                feature.append(n_neighbors[0][i] + gap*dif)
            feature[-1] = 1.0
            New_data.append(feature)
    return New_data
```

Algorithm 5: SMOTE

```
def _Smote_func(recombination, No_of_neighbours, Percentage):
    No_of_minority=len([i for i in (np.array(recombination)) if float(i[-1])
    == 1.0])
    No_New_instances = int((Percentage / 100) * No_of_minority)
    # Number of attributes excluding the class attribute.
    No_of_attributes = len(np.array(recombination)[0]) - 1
    Sythetic_values  = []

    for Pos_instance in [j for j,i in enumerate(np.array(recombination))
    if float(i[-1]) == 1.0]:

      k_distances=[[euclidean_distance(np.array(recombination)[Pos_instance
      ], np.array(recombination)[i]), i]
      for i in range(len(np.array(recombination))) if i != Pos_instance and
                      np.array(recombination)[i][-1] == 1.0]
        k_distances.sort(reverse = False)
        nnarray = [np.array(recombination)[j] for j in [i[1] for i in
        k_distances[0:No_of_neighbours]]]

    for newIndex in range(No_New_instances):
        row = []
        nn = np.random.randint(0, No_of_neighbours)
        for attr in range(0, No_of_attributes):
            difference=float(nnarray[nn][attr])-
            float(np.array(recombination)[Pos_instance][attr])
            gap = np.random.uniform(0, 1)
            row.append(np.array(recombination)[Pos_instance][attr]+
            (difference * gap))
        row.append(float(1.0))
        Sythetic_values.append(row)
    return Sythetic_values
```

Algorithm 6: Modified SMOTE

```
def _Smote_func_new(recombination, No_of_neighbours, Percentage):

    No_of_minority   = len([i for i in (np.array(recombination))
    if float(i[-1]) == 1.0])
    No_New_instances = int((Percentage / 100) * No_of_minority)
    # Number of attributes excluding the class attribute.
    No_of_attributes = len(np.array(recombination)[0]) - 1
    Sythetic_values  = []

    for Pos_instance in [j for j,i in enumerate(np.array(recombination))
    if float(i[-1]) == 1.0]:

k_distances=[[euclidean_distance(np.array(recombination)[Pos_instance],
np.array(recombination)[i]), i]
    for i in range(len(np.array(recombination))) if i != Pos_instance and
                      np.array(recombination)[i][-1] == 1.0]
        k_distances.sort(reverse = False)
        nnarray = [np.array(recombination)[j] for j in [i[1]
        for i in k_distances[0:No_of_neighbours]]]

    for newIndex in range(No_New_instances):
        row = []
        nn = np.random.randint(0, No_of_neighbours)
        for attr in range(0,No_of_attributes):
            row.append(np.random.uniform(float(nnarray[nn][attr]),
            float(np.array(recombination)[Pos_instance][attr])))
        row.append(float(1.0))
        Sythetic_values.append(row)
    return Sythetic_values
```

Algorithm 7: Modified Safe Level SMOTE

```
def _safe_level_smote_new(recombination):

    New_data = []

    for Pos_instance in [j for j,i in enumerate(np.array(recombination)) if
float(i[-1]) == 1.0]:

        k_distances                                                    =
[[euclidean_distance(np.array(recombination)[Pos_instance],
np.array(recombination)[i]), i]
                      for i in range(len(np.array(recombination))) if i !=
Pos_instance]
        k_distances.sort(reverse = False)
        k  =  [np.array(recombination)[j]  for  j  in  [i[1]  for  i  in
k_distances[0:5]]]

        n = k.pop(np.random.randint(0,5))
        n_distances = [[euclidean_distance(n, np.array(recombination)[i]),
i] for i in range(len(np.array(recombination)))]
        n_distances.sort(reverse = False)
```

```
        n_neighbors = [np.array(recombination)[j] for j in [i[1] for i in
n_distances[1:6]]]

        SLp = len([i for i in k if i[-1] == 1])
        SLn = len([i for i in n_neighbors if i[-1] == 1])

        if SLn != 0:
            SL_ratio = SLp/SLn
        else:
            SL_ratio = np.inf

        if SL_ratio == np.inf and SLp == 0:
            continue
        else:
            feature  = []
            for i in range(len(np.array(recombination)[0])):
                if SL_ratio == np.inf and SLp != 0:
                    gap = 0
                elif SL_ratio == 1:
                    gap = np.random.uniform(0, 1)
                elif SL_ratio > 1:
                    gap = np.random.uniform(0, 1/SL_ratio)
                elif SL_ratio < 1:
                    gap = np.random.uniform(1 - SL_ratio, 1)

                feature.append(np.random.uniform(k[0][i],
n_neighbors[0][i]))

            feature[-1] = 1.0
            New_data.append(feature)

    return New_data
```

Algorithm 8: Univariate Selection Method

```
def Uni_select(Features, Class):
    test = SelectKBest(score_func = f_classif, k = 30)
    fit = test.fit(Features, Class)
    Score = list(fit.scores_)
    Univeriate_Scores = [(b,a)for a, b in zip(Features.columns, Score)]
    Univeriate_Scores.sort(reverse = True)
    return Univeriate_Scores
```

Algorithm 9: Recursive Feature Elimination Method

```
def Recursive(Features, Class):
    model = LogisticRegression()
    rfe = RFE(model, 1)
    fit = rfe.fit(Features, Class)
    ranking = fit.ranking_
    Recursive_Scores = [(b,a)for a, b in zip(Features.columns, ranking)]
    Recursive_Scores.sort()
    return Recursive_Scores
```

Algorithm 10: Feature Importance Method

```
def Importance(Features, Class):
```

```
    model = ExtraTreesClassifier()
    model.fit(Features, Class)
    importance = model.feature_importances_
    Importance_Scores = [(b,a)for a, b in zip(Features.columns, importance)]
    Importance_Scores.sort(reverse = True)
    return Importance_Scores
```

Algorithm 11: Artificial Neural Network (ANN)

Table 8.1: Artificial Neural Network class properties.

Function Name	Description
1) _init_weight	The function to initialize the weights.
2) _add_bias_unit	The function to add the bias unit value of 1 to the layers of the model.
3) _forward	The forward propagation function of the model.
4) sigmoid_prime	The derivative function of logistic function.
5) _backward	The backpropagation function.
6) L2_reg	The regularize term function.
7) _error	The model's error function.
8) _backprop_step	The function defined to update weights of the model.

The class of the ANN method is defined as follows:

```
class NNClassifier():

    #Constructor
    def __init__(self, n_classes, n_features, n_hidden_units=30,
                 l1=0.0, l2=0.05, epochs=200, learning_rate=0.01,
                 n_batches=1, random_seed=None):
        if random_seed:
            np.random.seed(random_seed)
        self.n_classes = n_classes
        self.n_features = n_features
        self.n_hidden_units = n_hidden_units
        self.w1, self.w2 = self._init_weights()
        self.l1 = l1
        self.l2 = l2
        self.epochs = epochs
        self.learning_rate = learning_rate
        self.n_batches = n_batches

    #Initialize Weights
    def _init_weights(self):
        w1 = np.random.uniform(-1.0, 1.0, size=self.n_hidden_units *
(self.n_features + 1))
        w1 = w1.reshape(self.n_hidden_units, self.n_features + 1)
        w2 = np.random.uniform(-1.0, 1.0, size=self.n_classes *
(self.n_hidden_units + 1))
        w2 = w2.reshape(self.n_classes, self.n_hidden_units + 1)
```

```
        return w1, w2

    # Adding Bias Terms
    def _add_bias_unit(self, X, how='column'):
        if how == 'column':
            X_new = np.ones((X.shape[0], X.shape[1] + 1))
            X_new[:, 1:] = X
        elif how == 'row':
            X_new = np.ones((X.shape[0] + 1, X.shape[1]))
            X_new[1:, :] = X
        return X_new

    # Sigmoid Fuction
    def sigmoid(self, activation):
        return 1.0 / (1.0 + np.exp(-activation))

    #Forward Feed
    def _forward(self, X):
        net_input = self._add_bias_unit(X, how='column')
        net_hidden = self.w1.dot(net_input.T)
        act_hidden = self.sigmoid(net_hidden)
        act_hidden = self._add_bias_unit(act_hidden, how='row')
        net_out = self.w2.dot(act_hidden)
        act_out = self.sigmoid(net_out)
        return net_input, net_hidden, act_hidden, net_out, act_out

    #Derivative of Sigmoid Function
    def sigmoid_prime(self, output):
        return output * (1.0 - output)
    #Backpropagation
    def _backward(self, net_input, net_hidden, act_hidden, act_out, y):
        y = list(zip(*y))
        sigma3 = act_out - y
        net_hidden = self._add_bias_unit(net_hidden, how='row')
        sigma2 = self.w2.T.dot(sigma3) * self.sigmoid_prime(net_hidden)
        sigma2 = sigma2[1:, :]
        grad1 = sigma2.dot(net_input)
        grad2 = sigma3.dot(act_hidden.T)
        return grad1, grad2

    #Regularizer
    def L2_reg(self, lambd, w1, w2, m):
        return (np.sum(np.square(self.w1)) +
np.sum(np.square(self.w2)))*(lambd/(2*m))

    #Loss/Cost Fuction
    def cross_entropy(self, output, y):
        SumEntropy = 0
        for ind in range(len(y)):
            if y[ind][1] == 1.0:
                SumEntropy += (-np.log(output[1][ind]))
            else:
                SumEntropy += (-np.log(1 - output[0][ind]))
        return SumEntropy

    #Error Rate
    def _error(self, y, output):
```

```
        L1_term = self.L1_reg(self.l1, self.w1, self.w2, len(y))
        L2_term = self.L2_reg(self.l2, self.w1, self.w2, len(y))
        error = self.cross_entropy(output, y) + L2_term
        return 0.5 * np.mean(error)

    #Updating Weights
    def _backprop_step(self, X, y):
        net_input, net_hidden, act_hidden, net_out, act_out =
self._forward(X)
        grad1, grad2 = self._backward(net_input, net_hidden, act_hidden,
act_out, y)
        grad1[:, 1:] += (self.w1[:, 1:] * (self.l2))
        grad2[:, 1:] += (self.w2[:, 1:] * (self.l2))
        error = self._error(y, act_out)
        return error, grad1, grad2

    #Predictions
    def predict(self, X):
        Xt = X.copy()
        net_input, net_hidden, act_hidden, net_out, act_out =
self._forward(Xt)
        return [np.argmax(ind) for ind in net_out.T]

    #Decision Making Function
    def softmax(self, inputs):
        return np.exp(inputs) / float(sum(np.exp(inputs)))

    #Class Probability Predictions
    def predict_proba(self, X):
        Xt = X.copy()
        net_input, net_hidden, act_hidden, net_out, act_out =
self._forward(Xt)
        return [self.softmax(ind) for ind in net_out.T]

    #Class with Highest Probability
    def probability(self, prediction_list, prediction_proba_list):
        proba = []
        for Index, Value in enumerate(prediction_list):
            proba.append(prediction_proba_list[Index][Value])
        return proba

    #Learn
    def fit(self, X, y):
        y = to_categorical(y, self.n_classes)
        X_data, y_data = X.copy(), y.copy()
        for epoch in range(self.epochs):
            error, grad1, grad2 = self._backprop_step(X_data, y_data)
            self.w1 -= (self.learning_rate * grad1)
            self.w2 -= (self.learning_rate * grad2)
            print('>epoch=%d, error=%.3f'%(epoch, error))
        return self

    # Score
    def score(self, X, y):
        y_hat = self.predict(X)
        return np.sum(y == y_hat, axis=0) / float(X.shape[0])
```

Algorithm 12: Support Vector Machine (SVM)

Table 8.2: Support Vector Machine class properties.

Function Name	Description
1) _init_weights	The initialization of the weights
2) _add_bias_unit	The function to add the bias unit value of 1 to the layers of the model.
3) _format	The function that formats the output values to float numbers.
4) _predict_proba	The function that predicts the class probability.
5) _predict	The function that predicts the output of the transaction.
6) _score	The function to output the score.
7) probability	The function to output the class probabilities.
8) _train	The function that allows the model to learn from historical data

The class of the SVM method is defined as follows:

```
class Support_Vaector_Machine():

    #Constructor
    def __init__(self, n_classes, n_features, learning_rate=0.01,
                 epochs = 100, random_seed=None):
        if random_seed:
                np.random.seed(random_seed)
        self.n_classes      = n_classes
        self.n_features     = n_features
        self.learning_rate  = learning_rate
        self.epochs         = epochs
        self.W              = self._init_weights()

    #Initialize Weights
    def _init_weights(self):
        W = np.random.uniform(0.0, size = self.n_classes * (self.n_features
+ 1))
        W = W.reshape(self.n_classes, self.n_features + 1)
        return W

    #Adding Bias Terms
    def _add_bias_unit(self, X):
        X_new = np.ones((X.shape[0], X.shape[1] + 1))
        X_new[:, 1:] = X
        return X_new

    #Format Output Values to Float Numbers
    def _formatY(self, y):
        group = lambda List, num: zip(*[List[i::num] for i in range(num)])
        y = to_categorical(y, 2)
```

```
        y = [ -1.0 if value == 0.0 else value for values in y for value in
values]
        y = [np.array(value) for value in list(group(y, 2))]
        return y

    #Probability of the Output
    def _predict_proba(self, X, y):
        net_input = self._add_bias_unit(X)
        Weighted  = self.W.dot(net_input.T)
        format_y  = self._formatY(y)
        X_pred    = Weighted.T * format_y
        return X_pred

    #Predictions
    def _predict(self, X, y):
        proba = self._predict_proba(X, y)
        pred = [np.argmax(value) for value in proba]
        return pred

    #Score
    def _score(self, X, y):
        pred = self._predict(X, y)
        Count = 0
        for i in range(len(pred)):
            if pred[i] == list(y)[i]:
                Count += 1
        return Count/len(y)

    #Class with Highest Probability
    def probability(self, pred_list, pred_proba_list):
        proba = []
        for Index, Value in enumerate(pred_list):
            proba.append(pred_proba_list[Index][Value])
        return proba

    #Train
    def _train(self, x_train, y_train):
        for epoch in range(self.epochs):
            pred = self._predict_proba(x_train, y_train)
            for ind in range(len(pred)):
                if (pred[ind][1] >= 0):
                    self.W = self.W + self.learning_rate * ((-2) *
(1/self.epochs) * self.W)
                else:
                    self.W = self.W + self.learning_rate *
((np.array(y_train)[ind]
                              * np.array(self._add_bias_unit(x_train))[ind])
                              - (2 * (1/self.epochs) * self.W))
        return self
```

Algorithm 13: K-Nearest Neighbour (KNN)

Table 8.3: k-Nearest Neighbour class properties.

Function Name	Description

1) _distance	The function that calculates the difference between the two values.
2) _euclideanDistance	The function to measure the gap size between two data observations.
3) _getNeighbors	The function to get the k-neighbours.
4) _predict	The function to predict the output of each transaction.
5) _probability	The function to calculate the class probabilities of transactions.
6) _getAccuracy	The function to return the accuracy of the model.

The class of the KNN method is defined as follows:

```
class k_Nearest_Neighbor():

    #Constructor
    def __init__(self, row_length, neighbors, TrainDataX, TrainDataY,
random_seed=None):
        if random_seed:
                np.random.seed(random_seed)
        self.TrainDataX  = TrainDataX
        self.TrainDataY  = TrainDataY
        self.row_length  = row_length
        self.neighbors   = neighbors

     #Distance Function
    def _distance(self, row1_value, row2_value):
        Distance = pow((row1_value - row2_value), 2)
        return Distance

    #Euclidean Distance Function
    def _euclideanDistance(self, row1, row2):
        distance = 0
        for ind in range(self.row_length):
            distance += self._distance(row1[ind], row2[ind])
        return np.sqrt(distance)

     #Function to get k nearest neighbors
    def _getNeighbors(self, TestDataX):
        k_nearest_neighbors = []
        for ind in range(len(self.TrainDataX)):
            dist = self._euclideanDistance(np.array(self.TrainDataX)[ind],
TestDataX)
            k_nearest_neighbors.append((ind, dist))
        k_nearest_neighbors.sort(key = operator.itemgetter(1),
reverse=False)
        return [k_nearest_neighbors[i] for i in range(5)]

     #Function for Prediction of transaction
    def _predict(self, TestRow):
        k_nearest_neighbors = self._getNeighbors(TestRow)
        for Neighbor in k_nearest_neighbors:
```

```
            count0, count1 = 0, 0
            if self.TrainDataY[Neighbor[0]] == 0.0:
                count0 += 1
            elif self.TrainDataY[Neighbor[0]] == 1.0:
                count1 += 1
        if count0 > count1:
            predictions = 0.0
        else:
            predictions = 1.0
        return predictions

     #Function to get predictions
    def _getPredictions(self, TestDataX):
        count = 0
        predictions = []
        for TestRow in np.array(TestDataX):
            predictions.append(self._predict(TestRow))
            print(count)
            count += 1
        return predictions

     # Class Probability Function
    def _probability(self, TestDataX):
        Percentage = []
        for TestRow in TestDataX:
            k_nearest_neighbors = self._getNeighbors(TestRow)
            count0, count1 = 0, 0
            for Index, Neighbor in k_nearest_neighbors:
                if self.TrainDataY[Index] == 0.0:
                    count0 += 1
                elif self.TrainDataY[Index] == 1.0:
                    count1 += 1
            if count0 > count1:
                Percentage.append(count0/5)
            else:
                Percentage.append(count1/5)
        return Percentage

     # Function to get the Accuracy
    def _getAccuracy(self, TestDataY, predictions):
        correct = 0
        for y in range(len(np.array(TestDataY))):
            if np.array(TestDataY)[y] == predictions[y]:
                correct += 1
        return (correct/float(len(np.array(TestDataY)))) * 100.0
```

Algorithm 14: Naïve Bayesian (NB)

Table 8.4: Naive Bayesian class properties.

Function Name	Description
1) Mean	The function to calculate the mean.
2) Standard Deviation	The function to calculate the Standard Deviation.

3) _calculateProbability	Gaussian/Normal Distribution function that calculates the class probabilities.
4) _summaize	The function that computes the standard deviation and the mean of features.
5) _separateByClass	The function to separate transactions by class values.
6) _summarizeByClass	The function to compute the mean and standard deviation of transactions separated by class values.
7) _CalculateClassProbabilities	The function that calculate the class probability.
8) _predict	The function to predict the output of each transaction.
9) _getPredictions	The function to get the list of predictions
10) probability	The function that returns the list of probabilities.
11) _getAccuracy	The function that returns the accuracy of the model.

The class of the Naïve Bayesian method is defined as follows:

```
class Naive_Bayesian():

#Constuctor
    def __init__(self, x, y, random_seed=None):
        if random_seed:
                np.random.seed(random_seed)
        self.TrainDataX = x
        self.TrainDataY = y

    #Mean Function
    def _mean(self, numbers):
        return sum(numbers)/float(len(numbers))

    #Standard Deviation Function
    def _stdev(self, numbers):
        avg = self._mean(numbers)
        variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-
1)
        return np.sqrt(variance)

    #Gaussian/Normal Distribution Function
    def _calculateProbability(self, RowValue, mean, stdev):
        exponent = np.exp(-(np.power(RowValue-
mean,2)/(2*np.power(stdev,2))))
        return (1 / (np.sqrt(2*np.pi) * stdev)) * exponent

    #Mean and standard deviation of features
    def _summarize(self,TrainData):
        summaries = [(self._mean(attribute), self._stdev(attribute))
                     for attribute in zip(*np.array(TrainData))]
```

```
        return summaries

    #Separated Transactions by Class Values
    def _separateByClass(self):
        separated = {}
        for ind in range(len(np.array(self.TrainDataX))):
            vector = np.array(self.TrainDataX)[ind]
            if (np.array(self.TrainDataY)[ind] not in separated):
                separated[np.array(self.TrainDataY)[ind]] = []
            separated[np.array(self.TrainDataY)[ind]].append(vector)
        return separated

    #Mean and Standard Deviation of Features by class values
    def _summarizeByClass(self):
        separated = self._separateByClass()
        summaries = {}
        for classValue, instances in separated.items():
            summaries[classValue] = self._summarize(instances)
        return summaries

    #Function to Calculate Class Probabilities
    def _calculateClassProbabilities(self, TestDataInstance):
        probabilities = {}
        summaries = self._summarizeByClass()
        for classValue, classSummaries in summaries.items():
            probabilities[classValue] = 1
            for ind in range(len(classSummaries)):
                mean, stdev = classSummaries[ind]
                RowValue = TestDataInstance[ind]
                probabilities[classValue] *=
self._calculateProbability(RowValue, mean, stdev)
        return probabilities

    #Function to Predict the output of Transaction
    def _predict(self, TestDataInstance):
        probabilities = self._calculateClassProbabilities(TestDataInstance)
        bestLabel, bestProb = None, -1
        for classValue, probability in probabilities.items():
            if bestLabel is None or probability > bestProb:
                bestProb = probability
                bestLabel = classValue
        return bestLabel

    #Function to get Predictions
    def _getPredictions(self,TestDataX):
        predictions = []
        for i in range(len(np.array(TestDataX))):
            result = self._predict(np.array(TestDataX)[i])
            predictions.append(result)
        return predictions

    #Function that Returns the Probabilities
    def probability(self, TestDataX):
        probas = []
        for TestDataInstance in np.array(TestDataX):
            proba =
list(self._calculateClassProbabilities(TestDataInstance).values())
```

```
            probas.append(proba[np.argmax(proba)])
        return probas

    #Function to get Accuracies
    def _getAccuracy(self, TestDataY, predictions):
        correct = 0
        for ind in range(len(TestDataY)):
            if TestDataY[ind] == predictions[ind]:
                correct += 1
        return (correct/float(len(TestDataY)))*100.0
```

Algorithm 15: Decision Tree (DT)

Table 8.5: Decision Tree class properties.

Function Name	Description
1) _train_using_entropy	The function is used to train the DT model.
2) _pred_proba	The function that calculates the class probabilities.
3) _prediction	The function that returns the list of predictions.
4) probability	The function that returns the list of the probabilities of predicted values.
5) _getAccuracy	The function to return the accuracy of the DT model.

The class of the Decision Tree method is defined as follows:

```
from sklearn.tree import DecisionTreeClassifier

class Decision_tree():

    def __init__(self, Gini_Criterion, Max_depth, Min_samples_leaf,
                 Entropy_Criterion, random_seed=None):
        if random_seed:
                np.random.seed(random_seed)
        self.Gini_Criterion = Gini_Criterion
        self.Max_depth = Max_depth
        self.Min_samples_leaf = Min_samples_leaf
        self.Entropy_Criterion = Entropy_Criterion

    def _train_using_gini(self, X_train, y_train):
        clf_gini = DecisionTreeClassifier(criterion = self.Gini_Criterion,
                              random_state = 1, max_depth = self.Max_depth,
                              min_samples_leaf = self.Min_samples_leaf)
        clf_gini.fit(X_train, y_train)
        return clf_gini

    def _train_using_entropy(self, X_train, y_train):
        clf_entropy = DecisionTreeClassifier(
                criterion = self.Entropy_Criterion,
                random_state = 1,
                max_depth = self.Max_depth,
```

```
                min_samples_leaf = self.Min_samples_leaf)
        clf_entropy.fit(X_train, y_train)
        return clf_entropy

    def _pred_proba(self, X_test, clf_object):
        predict_prob = clf_object.predict_proba(X_test)
        return predict_prob

    def _prediction(self, X_test, clf_object):
        y_pred = clf_object.predict(X_test)
        return y_pred

    def probability(self, pred_list, pred_proba_list):
        proba = []
        for Index, Value in enumerate(pred_list):
            proba.append(pred_proba_list[Index][Value])
        return proba

    def _getAccuracy(self, yTest, predictions):
            correct = 0
            for y in range(len(yTest)):
                if yTest[y] == predictions[y]:
                    correct += 1
            return (correct/float(len(yTest))) * 100.0
```

Algorithm 16: Stacking Ensemble Method

Table 8.6: Stacking Ensemble class properties.

Function Name	Description
1) Mode_Fun	The class property used for voting per transaction.
2) Vote_Function	The class property used for returning the list of voted transactions.
3) getAccuracy	The class property used for computing the accuracy of the model.
4) Unmatched_Indexes	The class property used for returning the count of misclassified credit card transactions.

```
class Voting():

    def __init__(self):
        pass

    def Mode_Fun(self, List):
        Predict, Count0, Count1 = 0,0,0
        for Value in List:
            if Value == 0.0:
                Count0 += 1
            elif Value == 1.0:
                Count1 += 1
        if Count0 > Count1:
            Predict = 0.0
```

```
        else:
            Predict = 1.0
        return Predict

    def Vote_func(self, Model_pred_A):
        predictions = []
        for Row in Model_pred_A:
            Mode = self.Mode_Fun(Row)
            predictions.append(Mode)
        return predictions

    def getAccuracy(self, TestDataY, predictions):
        correct = 0
        for ind in range(len(TestDataY)):
            if TestDataY[ind] == predictions[ind]:
                correct += 1
        return (correct/float(len(TestDataY)))*100.0

    def Unmatched_Indexes(self, TestDataY, predictions):
        incorrect = []
        for ind in range(len(TestDataY)):
            if TestDataY[ind] != predictions[ind]:
                incorrect.append(ind)
        return incorrect
```

Algorithm 17: Differential Evolution Optimization Method

```
class DifferentialEvolution():
    def __init__(self, TrainDataX, population_size, Scaling_Factor,
Max_Iterator, Crossover_Probability,random_seed=None):
        if random_seed:
                np.random.seed(random_seed)
        self.TrainDataX = np.array(TrainDataX)
        self.population_size = population_size
        self.Scaling_Factor = Scaling_Factor
        self.Max_Iterator = Max_Iterator
        self.Population = self.Initialize(self.population_size)
        self.Crossover_Probability = Crossover_Probability

    def Bounds(self):
        bounds = []
        for Feature in zip(*np.array(self.TrainDataX)):
            Min, Max = min(Feature), max(Feature)
            bounds.append([Min, Max])
        return bounds

    def Ensure_Bound(self, Vector, Bound):
        New_Vector = []
        for ind in range(len(Vector)):
            if Bound[ind][0] >= Vector[ind]:
                New_Vector.append(Bound[ind][0])
            if Bound[ind][1] <= Vector[ind]:
                New_Vector.append(Bound[ind][1])
            if Bound[ind][0] <= Vector[ind] and Vector[ind] <=
Bound[ind][1]:
                New_Vector.append(Vector[ind])
```

```
        return New_Vector

    def Cost_Func(self, x):
        sums = sum([x[i]**2 for i in range(len(x))])
        return sums

    # Target Individuals
    def Initialize(self, pop_size):
        bounds = self.Bounds()
        Population = []
        for Index in range(pop_size):
            Individual = []
            for Value1, Value2 in bounds:
                Individual.append(np.random.uniform(Value1, Value2))
            Population.append(Individual)
        return Population

    def RandomValues(self, LengthList):
        count = 0
        RandomList   = []
        while count < LengthList:
            RandomNumber = np.random.randint(0, self.population_size)
            if RandomNumber not in RandomList:
                RandomList.append(RandomNumber)
                count += 1
            else:
                count += 0
        return RandomList

    def Mutation(self):
        LengthList   = 3
        RandomList = self.RandomValues(LengthList)
        X1 = self.Population[RandomList[0]]
        X2 = self.Population[RandomList[1]]
        X3 = self.Population[RandomList[2]]
        diff = [x1 - x2 for x1, x2 in zip(X2, X3)]
        V_donor = [x1 + self.Scaling_Factor*diff for x1, diff in zip(X1,
diff)]
        V_donor = self.Ensure_Bound(V_donor, self.Bounds())
        return V_donor

    def Crossover(self,):
        V_Targets = self.Population
        V_donor = self.Mutation()
        V_trials = []
        for V_Target in V_Targets:
            V_trial = []
            for Index in range(len(V_Target)):
                crossover = np.random.uniform(0,1) # Check the interval
                if crossover < self.Crossover_Probability:
                    V_trial.append(V_donor[Index])
                else:
                    V_trial.append(V_Target[Index])
            V_trials.append(V_trial)
```

```
        return V_trials

    def Scores(self):
        Gen_Scores = []
        V_trials  = self.Crossover()
        V_Targets = self.Population
        for i in range(1, self.Max_Iterator + 1):
            Trial_Costs = [self.Cost_Func(V_trial) for V_trial in V_trials]
            Target_Costs = [self.Cost_Func(V_Target) for V_Target in
V_Targets]
            for Index in range(len(Trial_Costs)):
                if Trial_Costs[Index] < Target_Costs[Index]:
                    self.Population[Index] = V_trials[Index]
                    Gen_Scores.append(Trial_Costs[Index])
                else:
                    Gen_Scores.append(Target_Costs[Index])
        return Gen_Scores

    def Best_Population(self):
        Gen_Scores = self.Scores()
        Gen_min = min(Gen_Scores)
        Gen_Sol = self.Population[Gen_Scores.index(min(Gen_Scores))]
        print('Best Generation: ', Gen_min)
        print('Best Solution  : ', Gen_Sol)
        return Gen_Sol

    def sigmoid(self,activation):
        return 1.0 / (1.0 + np.exp(-activation))

    def getPredictions(self, Model_Probability_df):
        List = []
        Best_Solution = self.Best_Population()
        for Observation in np.array(Model_Probability_df):
            DotProduct =
self.sigmoid(Observation.dot(np.array(Best_Solution).T))
            if DotProduct >=  0.838:
                List.append(1)
            else:
                List.append(0)
        return List

    def getAccuracy(self, TestDataY, predictions):
        correct = 0
        for ind in range(len(TestDataY)):
            if TestDataY[ind] == predictions[ind]:
                correct += 1
        return (correct/float(len(TestDataY)))*100.0
```

Algorithm 18: Confusion Matrix Method

```
def ConfusionMatrix(Model_pred_df, y_test):
    Pivot = []
    for value in Model_pred_df.columns:
        Column = Model_pred_df[value]
        TP, TN, FP, FN = 0, 0, 0, 0
```

```
        for ind in range(len(Column)):
            if np.array(y_test)[ind] == 1.0 and Column[ind] == 1.0:
                TP += 1
            elif np.array(y_test)[ind] == 1.0 and Column[ind] == 0.0:
                FN += 1
            elif np.array(y_test)[ind] == 0.0 and Column[ind] == 1.0:
                FP += 1
            elif np.array(y_test)[ind] == 0.0 and Column[ind] == 0.0:
                TN += 1
        Pivot.append([value, TP, TN, FP, FN])
    return pd.DataFrame(Pivot, columns = ['Model', 'TP', 'TN', 'FP', 'FN'])
```

Algorithm 19: Confusion Matrix Stats Method

```
def ConfusionMatrixStat(Model_pred_df, y_test):
    Pivot = []
    for value in Model_pred_df.columns:
        Column = Model_pred_df[value]
        TP, TN, FP, FN = 0, 0, 0, 0
        for ind in range(len(Column)):
            if np.array(y_test)[ind] == 1.0 and Column[ind] == 1.0:
                TP += 1
            elif np.array(y_test)[ind] == 1.0 and Column[ind] == 0.0:
                FN += 1
            elif np.array(y_test)[ind] == 0.0 and Column[ind] == 1.0:
                FP += 1
            elif np.array(y_test)[ind] == 0.0 and Column[ind] == 0.0:
                TN += 1
        Class0, Class1 = 0,0
        for Value in np.array(y_test):
            if Value == 0.0:
                Class0 += 1
            elif Value == 1.0:
                Class1 += 1
        Pivot.append([value, TP/Class1, TN/Class0, FP/Class1, FN/Class0])
    return pd.DataFrame(Pivot, columns = ['Model', 'TP', 'TN', 'FP', 'FN'])
```

Algorithm 20: Confusion Matrix for Stacking Ensemble Method

```
def ConfusionMatrixSE(Model_predictions, y_test):
    Pivot = []
    Column = Model_pred_df
    TP, TN, FP, FN = 0, 0, 0, 0
    for ind in range(len(Column)):
        if np.array(y_test)[ind] == 1.0 and Column[ind] == 1.0:
            TP += 1
        elif np.array(y_test)[ind] == 1.0 and Column[ind] == 0.0:
            FN += 1
        elif np.array(y_test)[ind] == 0.0 and Column[ind] == 1.0:
            FP += 1
        elif np.array(y_test)[ind] == 0.0 and Column[ind] == 0.0:
            TN += 1
    Pivot.append(['Stacking Ensemble', TP, TN, FP, FN])
    return pd.DataFrame(Pivot, columns = ['Model', 'TP', 'TN', 'FP', 'FN'])
```

Algorithm 21: Confusion Matrix Stats for Stacking Ensemble Method

```
def ConfusionMatrixStatSE(Model_pred_df, y_test):
    Pivot = []
    Column = Model_pred_df
    TP, TN, FP, FN = 0, 0, 0, 0
    for ind in range(len(Column)):
        if np.array(y_test)[ind] == 1.0 and Column[ind] == 1.0:
            TP += 1
        elif np.array(y_test)[ind] == 1.0 and Column[ind] == 0.0:
            FN += 1
        elif np.array(y_test)[ind] == 0.0 and Column[ind] == 1.0:
            FP += 1
        elif np.array(y_test)[ind] == 0.0 and Column[ind] == 0.0:
            TN += 1
    Class0, Class1 = 0,0
    for Value in np.array(y_test):
        if Value == 0.0:
            Class0 += 1
        elif Value == 1.0:
            Class1 += 1
    Pivot.append(['Stacking Ensemble Method', TP/Class1, TN/Class0,
FP/Class1, FN/Class0])
    return pd.DataFrame(Pivot, columns = ['Model', 'TP', 'TN', 'FP', 'FN'])
```

Algorithm 22: Confusion Matrix for Differential Evolution Method

```
def ConfusionMatrixDE(Model_pred_df, y_test):
    Pivot = []
    Column = Model_pred_df
    TP, TN, FP, FN = 0, 0, 0, 0
    for ind in range(len(Column)):
        if np.array(y_test)[ind] == 1.0 and Column[ind] == 1.0:
            TP += 1
        elif np.array(y_test)[ind] == 1.0 and Column[ind] == 0.0:
            FN += 1
        elif np.array(y_test)[ind] == 0.0 and Column[ind] == 1.0:
            FP += 1
        elif np.array(y_test)[ind] == 0.0 and Column[ind] == 0.0:
            TN += 1
    Pivot.append(['Differential Evolution', TP, TN, FP, FN])
    return pd.DataFrame(Pivot, columns = ['Model', 'TP', 'TN', 'FP', 'FN'])
```

Algorithm 23: Confusion Matrix Stats for Differential Evolution Method

```
def ConfusionMatrixStatDE(Model_pred_df, y_test):
    Pivot = []
    Column = Model_pred_df
    TP, TN, FP, FN = 0, 0, 0, 0
    for ind in range(len(Column)):
        if np.array(y_test)[ind] == 1.0 and Column[ind] == 1.0:
            TP += 1
        elif np.array(y_test)[ind] == 1.0 and Column[ind] == 0.0:
            FN += 1
        elif np.array(y_test)[ind] == 0.0 and Column[ind] == 1.0:
```

```
            FP += 1
        elif np.array(y_test)[ind] == 0.0 and Column[ind] == 0.0:
            TN += 1
    Class0, Class1 = 0,0
    for Value in np.array(y_test):
        if Value == 0.0:
            Class0 += 1
        elif Value == 1.0:
            Class1 += 1
    Pivot.append(['Differential Evolution', TP/Class1, TN/Class0,
FP/Class1, FN/Class0])
    return pd.DataFrame(Pivot, columns = ['Model', 'TP', 'TN', 'FP', 'FN'])
```