Final Report
CM3203 - One Semester Individual Project (40 Credits)

Credit Card Fraud Detection Using Novelty Detection Techniques.

Sara E Aldossary
C1558065

Supervised by : Dr Yuhua Li
Moderated by : Dr George Theodorakopoulos

School of Computer Science and Informatics
Cardiff University

A Final Year Project Submitted for the Degree of
BSc Computer Science with Security and Forensics.

10, May 2019

Abstract

Nowadays, card fraud is increasing due to the prevalence of modern technology. Thus, Automatic systems to detect and prevent against card fraud are a significant tool in the financial industrys battle against card crime. Machine learning and novelty detection techniques approaches are taken in Credit Card Fraud Detection field, since they are effective technologies and methodologies and easy to apply at the same time. It is used in order to reduce fraud activities.

The main aim of this project was to create a program that detects and identifies potentially fraudulent credit card transactions from a given data set, and evaluate its performance to be compared with other classifiers with evaluation metric. The program was trained with the given data set, based on the some of the most popular Machine Learning and Deep learning Classification algorithms, which are, Random Forest, Isolation Forest and Neural Networks algorithm. load Balancing and Feature Selection were maintained throughout the project, and it was implemented using Python programming language. However, there were no autonomous system that will be able to categorically define a transaction as fraud. The objective was to highlight those transactions that have a high probability of being fraudulent based on some criteria, known or otherwise learnt.

Acknowledgements

I would like to express my appreciation to the people who have supported me throughout this project. Firstly, I would like to thank my supervisor Dr. Yuhua Li for his support and guidance through out the semester. It was a pleasure working with him.

I would also like to thank my parents, *Essa Aldossary* and *Amal Aldossary* for the constant help and support they always provide, and for believing in my abilities to let me pursue my dream. Another thank you to my little sisters and brother who I dearly miss.

I will always be thankful for my beloved friends who were always there for me, for their constant support, and for the good memories we shared throughout this unforgettable journey.

Lastly, I would like to thank my sponsoring company *Saudi Aramco* for providing me with this great opportunity that helps me achieve my goals.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter consists of, a preface that includes a brief background of credit card fraud detection topic, followed by aims and scope of the project, motivation, the main intended beneficiaries, and a listing of the report structure.

1.1 Preface

Nowadays, card fraud is increasing due to the prevalence of modern technology. As [11] stated, card fraud losses total was around 567 million in 2015, and there was a 9% increase in 2016 with a loss of 618 million. Enterprises and public institutions must be well prepared to defend against such frauds that cause the loss of billions of dollars worldwide annually. Automatic systems to detect and prevent against card fraud are a significant tool in the financial industrys battle against card crime. However, it is not easy, or always possible to detect fraudulent patterns in transaction data by programmatic rules-based systems or inspection by fraud analysts; especially in large data sets. Credit card usage is increasing year on year, and the credit card and card payments market plays a huge role in todays economy. Predicted statistics by [32] indicated that credit card payments will increase over the next decade. The predicted transactions growth is to 3.7 billion in 2026, from 2.8 billion in 2016. Which consequently indicated that economic growth is one of the significant drivers of the credit card volumes and values future. Moreover, credit card usage has well known advantages such as; simplicity of payment, access to credit,

purchase guarantees and financial management to name a few, unfortunately these do not come without the risk of becoming a victim of fraudulent transactions. To reduce credit card fraud, it is important to use expert rules and statistical based models, such as rules based detection engines, machine learning and novelty detection techniques such as Clustering, Classification based and Nearest Neighbour, that can be used to distinguish potential and genuine fraud [22]. Thus, classification is the one will be mainly used in order to solve this problem so it will be discussed in further details later. Moreover, effective technologies and methodologies that can detect fraud and illegal activities such as; money laundering, have been delivered and applied earlier successfully as detailed here: `https://projecteuclid.org/download/pdf_1/euclid.ss/1042727940`
Machine learning algorithms have existing efficient fraudulent patterns (patterns will be discussed in detail at a later section), which typically found in data to be uncovered, or sometimes they do not, and predict possible fraudulent transactions, as it is a critical part of the fraud detection toolkit. A real or representative data set plays a significant role when it comes to build a system in machine learning as well. Specifically, to have a better fraud detection model there must be a large set of data to be used.

1.2 Project Aims and Scope

The main aims of this project are to establish a background in fraud detection novelty techniques and machine learning methodologies that will be used to complete this project, followed by detecting potentially fraudulent credit card transactions and classify both legitimate and fraudulent transactions from a given data set, where supervised and unsupervised Machine Learning Classification Algorithms and Novelty Detection Techniques are used on training the models. In the given data set, each individual transaction is already allocated to a different known class, either (1) for fraud or (0) for legit. Finally, offering discussion, comparison between the techniques, and evaluation of the solution produced.

However, it is important to remember during the whole project, that no autonomous system will be able to categorically define a transaction as fraud. The objective will be to highlight those transactions that have a high probability of being fraudulent based on some criteria, known or otherwise learnt.

Correspondingly, the scope of this project focuses on creating a well-functioned software system model depending on known behaviours or feature variables. It allows the beneficiaries to inspect the performance or the accuracy of the various machine learning classification algorithms which can classify the transactions in training and testing sets into either fraud or legit in real time. The data given is highly unbalanced data set, therefore, the approach used in this project is to implement different machine learning algorithms, which are: Random Forest as there are many cases where it has been used successfully, Isolation Forest and Neural Networks.

1.3 Motivation

Developing a system that detects credit card fraud detection is essential nowadays, and it is been done numerous times, like Pay360 integrated browser-based card payment solution [10]. These systems were created as a repercussion of customers preferring to pay with the most accepted payment mode which is the credit card, for example, in online shopping and paying their bills. Meanwhile, fraud transaction risks using credit card is also a critical issue that needs to be mitigated as far as possible. As there is no effective way to avoid these risks completely, there are ways to lower them by using various techniques of data mining needed by the companies along with fraud analysts and custom off the shelve solutions, to manage the data in real time. The main outcomes of this project are to absorb and understand some machine learning and deep learning techniques and show how it can be utilised in the field of credit card fraud detection, in order to provide better accuracy in this project along with avoiding computational complexity. Ensemble trees algorithms, and neural network which is one of the most commonly used Artificial Intelligence (AI) applications, are the two techniques used to overcome the previous challenges. Moreover, the accuracy will be compared between all the techniques used to select the best model, which will depend on the given data set. However, financial firms can benefit from machine learning techniques if a fraud detection system is installed, as it plays a huge rule in classifying fraudulent and legitimate transactions. If the project goals are achieved, it will prove that machine learning can be effective in mitigating the exposure of the credit card fraud for the card holder, the card issues as well as intermediating payment service providers. It may also result in discovering potential frauds earlier, and reducing the overall losses.

1.4 Intended Audience

The intended audience and beneficiaries of this project are the people doing or planning to be involved in a research in the field of fraud detection and security applications that are built using novelty techniques and machine learning methodologies, especially in the case of Credit Card Fraud Detection. The beneficiaries could be, fraud analysts, card issuers, payment service providers, academies in the area of fraud management and cardholders interested in how fraud can be reduced. However, credit card companies, specifically, must be able to recognize fraudulent credit card transactions by similar techniques, as it affects the customers by being charged for items that they did not purchase at all, along with the financial losses that affects the company itself.

1.5 Report Structure

The structure of this paper is organised as the following, **Chapter 2** illustrates an initial Background of Machine Learning, and its approach to Credit Card Fraud Detection, the Learning Evaluation process applied using evaluation performance metric, Python Libraries will be used in this project, and the Related Solutions includes related researches that have been conducted in the same area, **Chapter 3** demonstrates the Requirements Specification and Design of the software system, **Chapter 4** emphasis on the Implementation of the software system down to the code stage which includes the Structure of the Project, the Data Preparation, Classifiers used, and the Command-line Interface constructed, **Chapter 5** displays the test cases that were carried on for the System Evaluation , **Chapter 6** illustrates the Results and Discussion obtained from the implemented software system along with the Limitations of the approach taken, the potential Future Work that could be followed to improve the developed project, and the project Conclusion summarises the main findings and **Chapter 7** illustrates the Reflection on Learning gained from completing this project.

Chapter 2

Background Research

This chapter will briefly explain various concepts that were essential to understand before implementing the software system. The concepts that will be discussed are machine learning in general, outlier detection and feature selection and how it could be applied in machine learning, supervised and unsupervised machine learning. Also, the approach of machine learning in credit card fraud detection will be discussed along with the three classifiers that will be implemented. Finally, the learning evaluation using the confusion matrix and performance indicators will also be explained.

2.1 Machine Learning

Machine Learning tool has many roles in the industry of computing technologies since it was improved. Machine Learning term was defined in 1959, by Arthur Samuel as the Field of study that gives computers the ability to learn without being explicitly programmed [23]. A wide range of industries have realised the value of Machine Learning Technology while dealing with big data. It offers the organisation the opportunity to gain an advantage over other competitor industries, as different purposes have been utilised, such as; help detecting fraud and minimize identity theft for governments. Whereas in financial industry and banks, machine learning is utilised for two main reasons, which are, prevent fraud, and detect important insights in data. Some applied examples of machine learning are, search engine result refining, email spam and malware filtering, and credit card fraud detection [30]. The base of these applications is a combination of various disciplines, that are theory of algorithms, information theory and statistics and functional analysis.

In the 1950s, machine learning algorithms were deployed, and have major enhancement to be able to solve complex mathematical calculations to big data repetitively and faster than before. One important aspect of machine learning is the iteration, because the models are able to independently adjust, learn and train from previous computation as they exposed to new data [23].

Machines are significantly better than people, due to the speed of CPUs and computation cycles and the large storage capacity in processing large data sets. Moreover, machines have the ability to recognise and detect thousands of patterns in a single purchase by applying methods and technologies on raw data, instead of the rule-based techniques used by humans. Consequently, techniques used by Machine Learning are eye-catching in fraud detection field for several reasons. Speed is the first factor mentioned in [21], machine learning is able to examine large number of transactions in real time, analyse and process new data continuously, and it also has advanced models like neural networks which updates its models autonomously considering the latest trends. This cannot be comparable with the rule-based system as it is done manually and it is time consuming. Second factor is Scale, comparing to rule-base system, machine learning models become more effective with larger data sets. As the model obtains differences and similarities across transactions it potentially learns which transactions are fraud, and which are legitimate. It is also applicable to work with new data set, and the model can still be able to predict transactions. However, the only risk that scaling could cause is that the machine learning algorithm will train the system to ignore any undetected fraud found in the training data in the future. Machine learning is also efficient as it can perform repetitive tasks along with identifying fraudulent transaction by detecting indirect or non-intuitive patterns. False positives in confusion matrix can be avoided by using machine learning algorithms as well [21].

Machine Learning with fraud detection is an application in new and novel ways. It is more effective to join supervised and unsupervised methods in one system rather than using a single method by itself. It also referred to as anomalies and abnormalities.

2.1.1 Outlier Detection

An outlier can be described as an observation which varies a lot from other observations. However, fraud detection nowadays has been applied using techniques like neural networks, data mining, classification and clustering, etc. Outlier detection is a mechanism of looking in the data set for patterns that do not stratify to expected behaviour. Credit card fraud detection is a critical application of outlier detection along with several application domains as well as research fields, intrusion detection on cyber-security and homeland security, for instance. Outlier detection plays a huge rule as anomalies in data, as it can designate harmful or unfamiliar activities for different application areas. In credit card transaction data, outliers can be an indication of identity theft or credit card fraud. One of the many techniques that have been proposed in the past is classification based approaches in outlier detection. It works into two phases, training and testing phases. The training phase learns a classifier, whereas the testing phase classifies data instances. Since outlier detection considered as unsupervised data learning problem, any abnormal data instance added or removed will affect the principal direction than adding or removing a normal one. So, normally to determine the Outlierness of a specific data instance this strategy is used.

2.1.2 Feature Selection

A significant step of data pre-processing in machine learning applications is feature selection. It is mostly applied on features to find the smallest subset that extremely increases the model performance. It also involves other benefits, like the capability to construct faster and simpler models by using a single subset of the entire feature set. By focusing on a selected subset features, it also gains a better understanding of the process described by the data [29]. According to Saeys [29], there are three categories of feature selection techniques that relies on its interaction with a classifier. Filter methods provide a feature weighting, subset as output or ranking, and they also operate on the data set directly. The advantages of these methods are summarised as they are significantly fast and independent of the classification model, but at the price of inferior outcomes. Moreover, wrapper methods execute a search in feature subset field, guided by the model outcome, like the performance of a classification on cross-validation of the training set. Better results than filter methods are usually performed by wrapper methods, but it costs an increased computational price. Lastly, internal information of the classification model is used by embedded methods to perform the feature selection. A good balance between performance and computational cost is provided by embedded methods. The method utilised for the purpose of this project was the filter methods, as it considers the relationship between the transactions and its labels of fraudulent and legitimate to compute the importance features. The SelectKBest() algorithm using F-test will be used, f_classif specifically for classification tasks, while F-test according to Asaithambi [2] is a statistical test utilised in comparing models and checking how significant is the dissimilarities between the models . This is used in feature selection to know how significant each feature is in improving the model.

2.1.3 Supervised and Unsupervised Machine Learning

Known and unknown patterns extracted from the data are automated by machine learning, these patterns are expressed as a formula or instruction set that can be applied to unseen and new data. The more the outcomes and new patterns are given to a machine learning, the more it learns and adapts. Also, it can either be Supervised or Unsupervised Machine learning. The supervised machine learning is defined in [30] as a class contains analytic methods, it learns from labelled data that are usually represented by specific records in data. When the data is understood, based on specific pattern, the algorithm decides the labels that will be given to new data, and the patterns will be associated to unlabelled new data. Moreover, when training a supervised model, the records of both classes should be presented, for example fraudulent and legit transactions, so the model

attempt to deduce a function that is able to predict whether the transaction is fraud by applying it to new examples. It can be divided into classification and regression.
Classification algorithms, which is the main focus of this project, predict which category the data belongs to, whereas regression predicts a numeral value based on previously observed data. Commonly used supervised machine learning methods are neural networks, decision trees, and support vector machines, etc. [30].
However, unsupervised machine learning is where the algorithm learns from unlabelled or unclassified test data. It can be used to discover the underlying structure of the data, where it identifies what the data has in common, and reacts in each new piece of data based on the occurrence or absence of such commonalities, instead of responding to feedback. It also seems to reveal unknown patterns in data, but if the patterns are poor approximations of what the supervised machine learning can achieve, and that is the case most of the time [12]. Moreover, there is no way to determine how accurate are the outcomes if it is unknown, which makes the supervised algorithms more applicable to real-world problems as well. However, the unsupervised algorithms are better used when there is not data on the desired outcomes. Common techniques of unsupervised machine learning are clustering, anomaly detection and association mining, etc. [7].
In fraud detection, the fraudulent data is unknown, the model should create a function that labels the data structure, where any data that does not fit the model will be flagged as anomaly. unsupervised methods main challenge is, that it is usually hard to measure the detection scheme accuracy until the data is verified and worked out by hand [30].

2.2 Approach of Machine Learning to Credit Card Fraud Detection

Fraud detection is considered as a binary classification task, where in this project, any transaction will be predicted and identified as fraudulent transactions (1), and legitimate transactions (0).
Machine Learning algorithms and statistical approaches are utilised to construct classi-

fiers software, which is trained using labelled training data that consists of transactions and labels, and can be used later to map unseen data, as shown in figure 2.1 below. According to [12], number of machine learning algorithms have been used successfully for this problem, such as Random Forest and Neural networks, since traditional tools of fraud detection are insufficient nowadays. Isolation forest is another recommended method in the detection field, as mentioned in [20].

All of the above mentioned classifiers will be further discussed in the upcoming subsections.

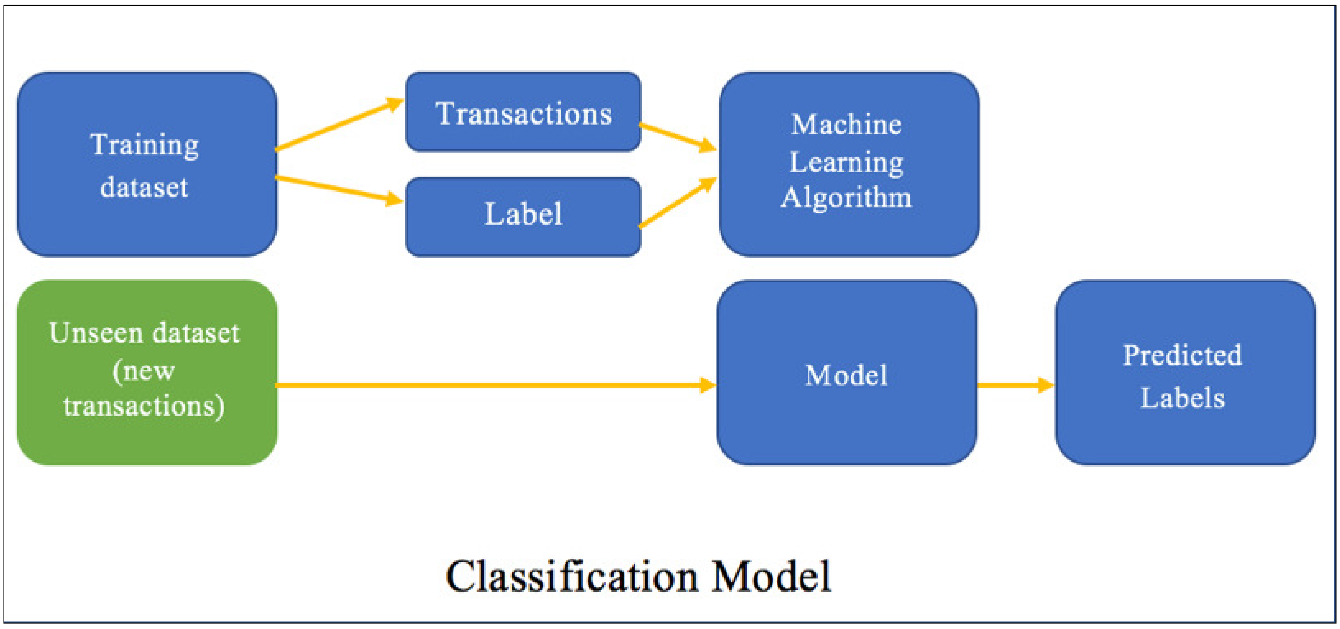


Figure 2.1: Classification model diagram

2.2.1 Random Forest

Random forest algorithm is an ensemble learning proposed in [3], as a supervised learning algorithm. Noting that the ensemble learning is referred to as the procedures employed to train multiple learning machines and combine their outputs, treating them as a committee of decision makers [5].

Moreover, according to [17] random forests algorithm consists of combining the estimate of a number of decision trees created. There is a direct relationship between the number of trees and the accuracy level. Also, there is an optimum number of trees that can generate high accuracy levels. This method is a predictive not descriptive modelling tool, fast, simple and flexible. Additionally, it can be used for both classification and regression problems as well as that it runs efficiently on large databases, and has methods that balance the error in class population of unbalanced data. It can also handle the different feature types such as; binary, numerical and categorical. However, in this project it will be used to solve a classification binary problem. Furthermore, it can handle missing values in two possible ways, either by using median values to replace the continuous, or by computing the missing values proximity-weighted average. Moreover, in random forest, to train each individual tree as well as building a decision tree, a subset of the training set is sampled in a random base. Also, each node splits on a selected feature from a random subset of the full feature set [17]. Similarly, for many features and data instances in large data sets, the training process in random forest is extremely fast. That is because each tree is trained separately from the others. The algorithm also provides a good generalisation error estimate along with its over fitting resistance [ibid]. Moreover, when evaluating the class votes of all bootstrap samples in a classification problem event, new observations are classified by random forest.

Hence, all leaves are spread around the decision tree built, involve an individual class, where the representative for the nth tree will be the highest predicted class. The final result will be specified by considering the majority vote of the whole forest (Tree-1, ,Tree-n) [3]. The process flow in random forest models is illustrated in figure 2.2 below.

Another considerable property of random forest is the ease of measuring relative importance of every feature on the prediction [8]. Feature_importances_ is a useful tool offered by Scikit-learn as it evaluates the importance of features on an artificial classification task, depending on tree nodes number that use this feature and decrease impurity within all the tree across the forest. Moreover, this score is computed automatically for each single feature, after training and scaling the results, thus, the total of all importance is equal to 1.

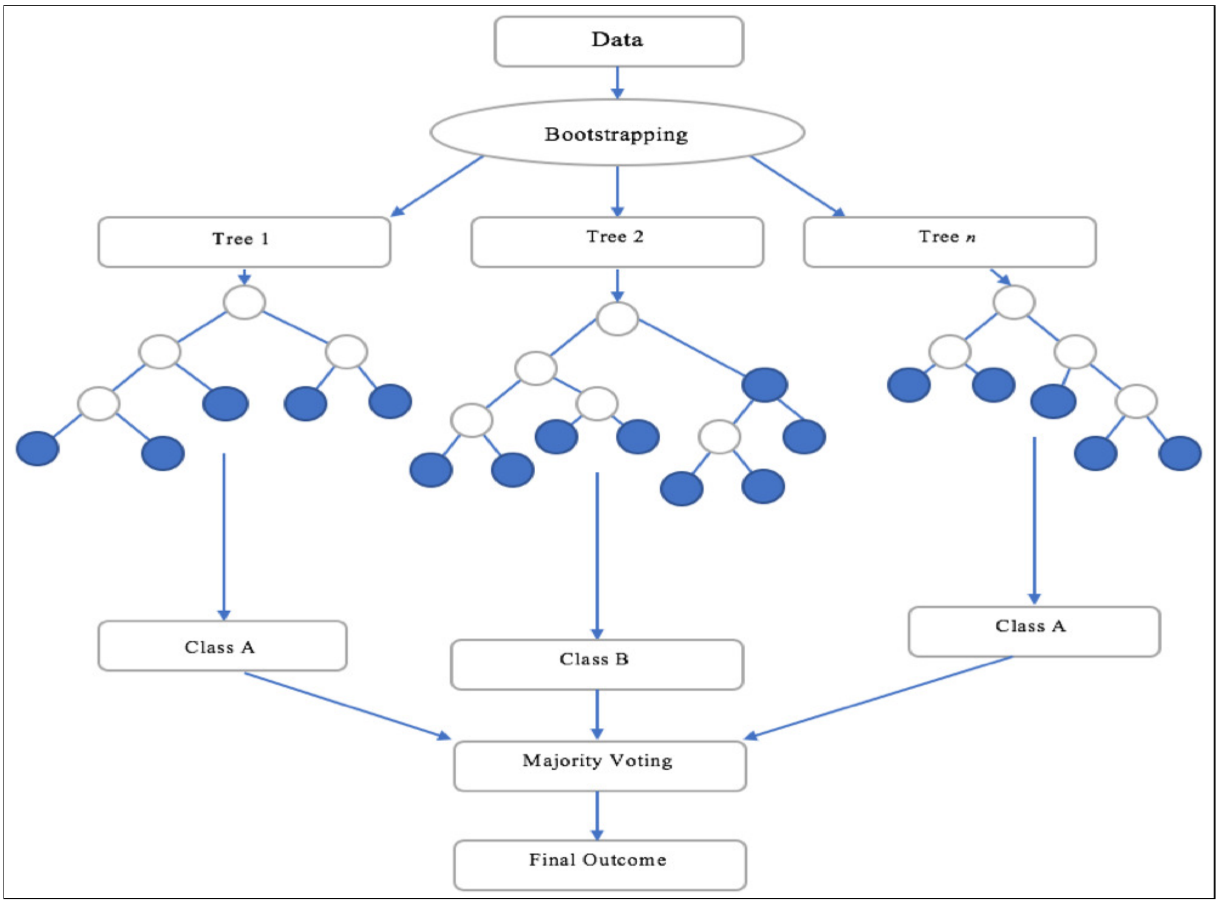


Figure 2.2: Classification model diagram

Feature importance helps in deciding which features must be dropped, as they either do not contribute enough or not contributing at all in the process [ibid]. Model over fitting could possibly occurs in some cases, when a large number of features is added. That obviously depends on the classifier and the data set are used, as the general rule stated. However, Random Forest has an advantage over decision tree classifier, that it corrects the over fitting to their training set habit, and it is also considered as a robust and a highly accurate method due to the number of trees participating [24].

2.2.2 Isolation Forest

Isolation Forest is an anomaly detection (unsupervised) algorithm developed by [20] as a tree-based model that detects outliers in machine learning-base approaches. Many applications are applying this technique such as, credit card transactions fraud detection, as it is one of the latest techniques in detecting anomalies, and it is constructed on the

fact that the anomalies are few and different data points. These properties resulted in anomalies to be vulnerable in the isolation algorithm. Isolation forest is highly useful and different from other algorithms, as isolation is used to detect anomalies in a more effective and efficient way than the basic distance and density measures. It also requires a small memory and a low linear time complexity, since it builds a good performing classifier with a small amount of trees, regardless of the size of the data set.
The Isolation Forest mechanism, consists of a group of isolation trees which are built for the same data, where the isolation tree is created by partitioning X recursively, where X is sample of data that has n instances from a d-variate distribution, with a random selection of a feature along with a random split value, which is between the maximum and minimum number of the selected feature, until either: one point only left in the node, all the data in the node has an identical value, or if a height limit of the tree is reached. Noticeable shorter paths for the anomalies are produced because of the random portioning happened. The decision predication requires formula of outlier detection of an anomaly score, in Isolation Forest, [1] defined it as:

$$s(x, n) = 2 - E(h(x))/c(n)$$

whereas, h(x) considered as observation x path length, and c(n) is the average of failed search in a binary search tree path length. And finally, n is the number of other nodes So, the path length of a point x (h(x)) is measured using x, the number of edges traverses an isolation tree from the root node until determining the traversal at an external node. If the number of trees is large enough, usually the average of h(x) from isolation forest converges [33].

2.2.3 Neural Networks

Neural networks are a set of algorithms that can simulate the learning algorithm of the human brain cells. They are a flexible computing framework that can be used to address pattern recognition and classification problems. Although that they are commonly used in Artificial Intelligence application and they are a deep learning application, they are also considered as a machine learning approach. As they interpret sensory data through a kind of machine perception, labelling or clustering raw input. However, the recognised patterns by neural networks are numerical, contained in vectors, into which all real-world data, text or sound must be translated [27]. Classifying and clustering tasks can be done using neural networks, whereas a clustering and classification layer on top of the stored and managed data. Unlabelled data can also be grouped according to similarities among the sample inputs, and when a labelled data set is available to train, the data will be classified.

Another benefit of neural networks is that it can also extract features that are given to other classification algorithms, which means that deep neural network can be as elements of larger machine learning applications including algorithms for classification, regression and reinforcement learning [27].

In contrast to some other machine learning approaches, such as classification models, artificial neural networks can capture connection or discover regularities within a set of variables. Moreover, the domain of applying neural network models frequently relies on the nature of the problem that will be modelled, as these systems are precisely suitable for domains that have dynamic and non-linear relationships. Generally, the most known situations addressed by neural networks in [9] are: firstly, when the number of data variables or its volume is varied. Also, when the connection or relationship between the variables is not easily identified or it is inherently complex, and when there is a demand for modelling several behaviours by discovering patterns among cases [ibid]. In an usual classification task, for instance, the task is to discover a function *f*, which satisfies

$$f(a) = b$$

So that this function f is capable to map an instance a to its class b.

However, artificial neural networks are attempting to discover a group of functions F, where

$$f1, f2, ..., fn \in F$$

and

$$fn(fn\text{-}1(\ldots(f2(f1(x)))\ldots))=f(a)=b$$

Instead of only finding the function *f* [13].

While, the list of *f1, f2, , fn* are linked in a chain to complete the task of function *f*, as it can be seen from the structure of artificial neural networks.

One of the approaches taken in developing a Neural Networks model is by using Keras [16] library, following four main steps shown in figure 2.3.

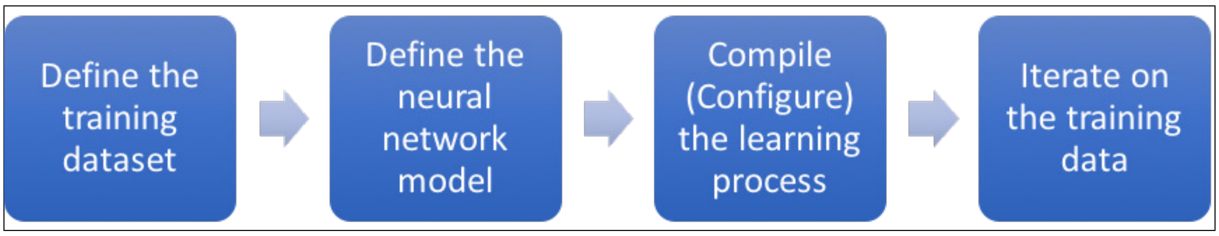

Define the training dataset
Define the neural network model
Compile (Configure) the learning process
Iterate on the training data

Figure 2.3: Classification model diagram

Defining the training data is the first step, and it is basically done by defining the input transactions alongside its labels, after selecting the features, cleaning the data from any null or missing values, and pre-processing the data as this is the crucial part in any machine learning task. The model at this stage does not typically concerns processing any hyper-parameters. Secondly, there are two models in defining a neural network model using Keras. This step is done either by using Sequential model class, or the Functional API, where both models take different approach and share the same goal at the same time, which is defining a neural network according to [4]. The Sequential model approach is utilised and limited in defining a linear stack of network layers, that will collectively constitute a model. However, the Functional API model approach provides the flexibility necessitated for more complex models [ibid]. The flexibility provided is best represented in the use cases of all the multi-input multi-output and the definition graph-like models. This report will follow the Sequential model approach with utilising the functions provided by Keras.

Additionally, the configuration learning process comes after defining the training data and the model. In order to complete the learning process configuration, compiling the

Sequential model class is the next step in this stage, requiring three compilation arguments, which are: the loss function, optimizer used and a list of metrics. The last stage of creating the neural network model, training the model. This step is made by passing the data into the model for training process. For the purpose of completing the training, the training data needs to be iterating after the training begins. This requires fitting at least two arguments which leads to a single iteration automatically performed, which typically does not work properly unless additional arguments are added.

2.3 Learning Evaluation

This section will include the evaluation techniques used in measuring the evaluation of each model.

2.3.1 Confusion Matrix

Various evaluation techniques approaches are available to measure the performance of a model. Some commonalities are shared even though they are different in how and what they measure, for example, confusion matrix is based on deriving values [35]. In the given data set, both legitimate and fraudulent transactions with classes (0) and (1), the most useful measurement is confusion matrix. It easy to explain in binary classification problems, and it is easy to interpret [ibid]. A collection of input objects *Xi,i=1,...,n* resulted from each classification can produce a confusion matrix. Confusion matrix outcomes are (TP) True Positive, (TN) True Negative (FP) False Positive, and (FN) False Negative, and they are outlined in terms of the project with their definition below,

- **TP**: the cases of correct classification of a positive class [ibid].
- **TN**: the cases of correct classification of a negative class [ibid].
- **FP**: the cases of incorrect classification of a positive class, that is actually negative [ibid].

- **FN**: the cases of incorrect classification of a negative class, that is actually positive [ibid].

Table of CM	**Predicted Positive**	**Predicted Negative**
Actual Positive	(TP) True Positive	(FN) False Negative
Actual Negative	(FP) False Positive	(TN) True Negative

Table 2.1: Confusion Matrix Table

Additionally, confusion Matrix is a $(n\ x\ n)$ matrix, where n represents the number of possible classes, and it is used to compare between the actual class and the predicted class evaluation examples [ibid]. It typically presents the overall result of the outcomes in a confusion matrix with two dimensions as seen in the Confusion Matrix Table 2.1, while both Positive and Negative classes are represented as follows, each column of the matrix contains the predicted class, whereas each row of the matrix contains the actual classes. Specifically, each entry represents the total occurrences of the test set for each actual and predicted classes in rows and columns respectively [ibid].

2.3.2 Performance Indicators

Performance indicators are extracted from the confusion matrix, so, the good indicators will rely on having small numbers, ideally zero-diagonal in the off-diagonal element as well as large numbers in the diagonal elements [35].
Identifying the behaviour of a model when in use, in an accurate representation requires the raw data constructed from the confusion matrix. Utilising different kinds of performance indicators will help to do so. Besides Accuracy, different crucial aspects for evaluating the performance of a classifier are represented by the most known indicators Precision and Recall [ibid]. Furthermore, the number of instances that should certainly be positive, and were predicted as positive is measured by recall. However, the percentage of true positives as a ratio overall cases that should have been true is measured by precision [ibid]. Accuracy, on the other hand, indicates the total percentage of instances that were predicted correctly, where all of the indicators has the best value as 1 and the worst as 0 [ibid]. The calculations of the three performance indicators are as follows,

- Recall = TP/(TP+FN)
- Precision = TP/(TP+FP)
- Accuracy = TP/(TP+TN+ FP+FN)

2.4 Python Libraries of Machine Learning

This section will involve a general description of the most significant tools that will be used over the development stages of the machine learning models.
This project will be implemented using Python programming language. So, a research on the libraries used in machine learning had to be conducted. Several open source Machine Learning libraries are provided by Python, that can help in the main areas of this project including data analysis process, prototyping and evaluation. Libraries offered by python are enormous, like Scikit-learn library, Keras, Numpy and Pandas. Scikit-learn, involves good documentation and combines a high performance along with the ease of use. Hence, it makes achieving machine learning in production system easier, it provides some useful methods in this field, like, measures performance methods. It can also be used for validating and splitting the data into training, testing sets randomly [31]. Keras is also a python open source library, it is focus is more on Deep Learning, which is considered as an aspect of machine learning. It allows fast and easy prototyping within its user-friendliness, extensibility and modularity. It also optimises the use of both CPU and GPU continuously with a faster performance [Kears 2015]. Numpy is another essential package used in scientific computing and it can ease the process of creating models using API by converting the Pandas data frames into arrays that are compatible with API. It is also useful in random number capabilities, and it integrates with various databases flawlessly [25]. Pandas -Python Data Analysis Library- provides high-level data structures and data analysis tools that are intuitive and easy to use for different methods. It provides performing time series analysis, and various inbuilt methods for combining and filtering data. Also, it allows quick attributes modification along with the essential data representation provided [26]. For the purpose of this project, Scikit-learn will be

used for both Random Forest and Isolation Forest algorithms as the data will be analysed is not very large, in addition, to develop a faster software system, Pandas will also be used to take data files, such as, CSV file in the case of this project, and creates a data frame similar to the statistical software tables as an object with rows and columns. In the neural networks part, Keras library will be used as it implements fast and easy models, specifically Sequential API that contains a linear stack of layers.

2.4.1 Pickle

In order to ensure that the experiments done are reproducible, and ensures that the input data has not been changed, pickle was used to save this data. It implements an important and powerful algorithm used in serializing and de-serialising object structure in Python. The process of converting a Python object hierarchy into a byte stream is called Pickiling. Whereas the inverse of this operation where a byte stream is transformed back to an object hierarchy is called Unpickling, but both procedures can also be referred to as Serialisation [28]. Regarding this thesis, the serialised format will be saved into a pickle file pkl, that can be loaded later to de-serialise the model in order to get deterministic results, which are useful to run the same experiment that previously classified, and utilise it in making new predictions.

2.5 Related Solutions

Credit card fraud detection is one of the popular domains in machine learning and data mining to study and work with. Enormous amount of studies and research were utilised to prevent any fraudulent actions. Also, the strength of machine learning and data mining techniques was demonstrated in most of the papers. One of the approaches regarding credit card fraud detection using novelty techniques was carried out in [14] where a real-life data set containing credit card transactions given by an international credit card operation has been used. The main aim of this paper was to detect credit card fraud by employing transactions aggregation strategy that relies on creating variables for a logistic regression model estimation. Another study was suggested in [6], about a new algorithm using an online questionnaire that represents the collected QRT data. The study mainly focused on training data and predicting new transactions, that are either fraudulent or legitimate, using QRT models that need to be developed and utilised by a support vector machine (SVM). Another research carried out in [18], proposes an algorithm that utilises two-stage structure alignment to group both the misused detection and anomaly detection. Moreover, a weighted random forest algorithm has been conducted in this research [34], was to compute the weight based on decision tree out-of-bag-error.

However, the approach of this thesis is to solve the problem by applying two machine learning approaches along with a deep learning approaches. Machine Learning approaches will provide two models which are Random Forest as a supervised algorithm, and Isolation Forest as unsupervised algorithm. In addition, the deep learning techniques approach will implement a neural network anomaly detection algorithm. As most of the research conducted in this field were concentrating on a specific area of the numerous machine learning algorithms that can be used, this paper will differ by mainly focus on comparing the performance of three given classifiers and choosing the best model produced, instead of implementing a single approach. Also, Random Forest algorithm was firstly chosen among other decision trees as it was proven in [19] that it performs very well in different fields, including the fraud detection, and it is also resistant to over fitting problem that must be avoided indeed.

Chapter 3

Specification and Design

For this project to be successfully constructed, and met the initial objectives at the same time, a sufficient amount of time had to be given. It was also important to identify and design the main components that need to be created. Consequently, this chapter of the report will clearly discuss the Software System to be developed. Software requirements specification and initial plan of the design will be reviewed. It will also mention the system design and the methodology will be followed in this project.

3.1 Software Requirement Specification

Every software system includes the software requirements specification as the base for the entire project. As it helps to ensure that the requirements are fulfilled, as well as minimising the overall development cost and time. It consists of descriptions of the software will be developed. For this project the Software Requirements Specification involves functional and non-functional requirements represents the essential implemented functions performed by the software in the technical part. As they are also useful to evaluate the success of the final result.

3.1.1 Functional Requirements

The most important functional requirements of the system are outlined below:

- The System must take new transaction data from the user as an input (argument) and classify them into legitimate or fraudulent.
- The client must provide labelled historic transactions to help develop this system.
- The system must provide three different classification algorithms to the user to create the model. The classifiers available are, Random Forest, Isolation Forest and Neural Networks.
- The classification must be run from a simple command line interface application, where the user will be asked to provide a location of the unclassified data in a csv format along with the classifier chosen.
- The system should create a new csv file with the classification data appended.

3.1.2 Non-Functional Requirements

The non-functional requirements of the system are listed below,

- *Usability*
  - The system will create a small number of models to be used for classification.
  - The above mentioned models will be trained with labelled historic data.
  - These models will be evaluated using a test set from the historic data.
- *Performance*
  - The model feature set will be evaluated with the appropriate feature selection algorithms, aiming to choose the minimum feature set that produces the maximum accuracy. A faster classification usually produced when less features are chosen.
- *Reliability*
  - The models will be tested with historic data, so that an idea of the accuracy ability of each model will be given, which is also an indication of reliability.

  - The reliability matrices will be provided are, accuracy, precision, recall and confusion matrices.

- *Re-usability*
  - The trained models will be saved in a serialized object format (pickle), so that they can be reused without the need for a repeated training process.
  - The code implementation of the different classification processes will be divided into a reasonable amount of sub packages and can be reusable in any other classification systems with minor modifications to be applied.
- *Size*
  - The size of the models implemented will approximately use similar sizes mentioned below, hence the system will have a small impact on the computer memory:
    - Random Forrest model: around 954 KB
    - Neural Network model: around 29 KB
    - Isolation forest model: around 672 KB
- *Software*
  - The system will be created supports the following software versions:
    - Python version: 3.5
    - Scikit Learn: 0.20.3
    - Numpy: 1.16.2
    - Keras: 2.24

3.2 System Plan

The produced system will be implemented based on the given data set from Kaggle.com [15]. The data set contains 492 fraud transactions out of 284,807 legit transactions. The system will mainly classify these transactions depends on its class, 0 for legitimate and 1 for fraudulent transactions. The programming platform that will be used in this project is Python. The system will be implemented as a main package called CCFD which stands for Credit Card Fraud Detection, it will contain six classes, Random_Forest, Isolation_Forest, Neural_Networks, Split_Data, Evaluation and Classification_app. The Split_Data class will handle the most significant part in the software system functionality, which is preparing the data for the classifiers. The Evaluation class has a crucial role in providing the user with the accuracy and evaluation metric of the classifier used to classify the transactions. The three following classes Random_Forest, Isolation_Forest, Neural_Networks will handle the implementation of Random Forest, Isolation Forest and neural network classifiers. The Classification_app is a simple command-line interface, where the user can choose one of the pre-trained models in order to classify new data.

3.3 System Design

A Unified Modelling Language (UML) is created to have a better understanding of the system. UML diagrams also provide the developer with a template as a guide used to construct the software system, and it present a visual representation of the software system and how it is intended to be. Therefore, this section will present diagrams that have been created and provided from UML below,

The class diagram shown below in figure 3.1 presents the interaction of different classes in the software system, and the internal structure of it.

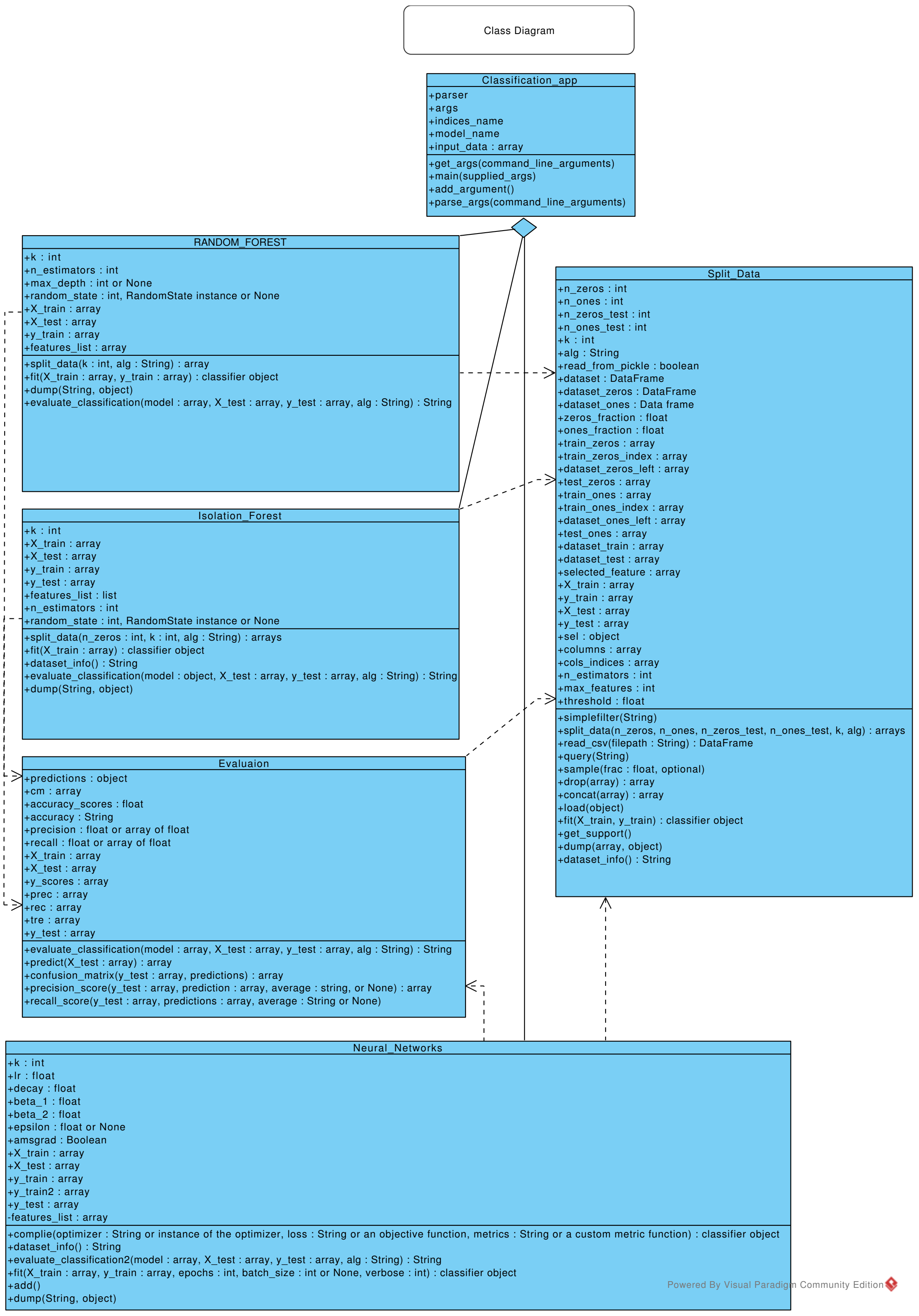

Class Diagram
Classification_app
+parser
+args
+indices_name
+model_name
+input_data : array
+get_args(command_line_arguments)
+main(supplied_args)
+add_argument()
+parse_args(command_line_arguments)
RANDOM_FOREST
+k : int
+n_estimators : int
+max_depth : int or None
+random_state : int, RandomState instance or None
+X_train : array
+X_test : array
+y_train : array
+features_list : array
+split_data(k : int, alg : String) : array
+fit(X_train : array, y_train : array) : classifier object
+dump(String, object)
+evaluate_classification(model : array, X_test : array, y_test : array, alg : String) : String
Isolation_Forest
+k : int
+X_train : array
+X_test : array
+y_train : array
+y_test : array
+features_list : list
+n_estimators : int
+random_state : int, RandomState instance or None
+split_data(n_zeros : int, k : int, alg : String) : arrays
+fit(X_train : array) : classifier object
+dataset_info() : String
+evaluate_classification(model : object, X_test : array, y_test : array, alg : String) : String
+dump(String, object)
Evaluaion
+predictions : object
+cm : array
+accuracy_scores : float
+accuracy : String
+precision : float or array of float
+recall : float or array of float
+X_train : array
+X_test : array
+y_scores : array
+prec : array
+rec : array
+tre : array
+y_test : array
+evaluate_classification(model : array, X_test : array, y_test : array, alg : String) : String
+predict(X_test : array) : array
+confusion_matrix(y_test : array, predictions) : array
+precision_score(y_test : array, prediction : array, average : string, or None) : array
+recall_score(y_test : array, predictions : array, average : String or None)
Split_Data
+n_zeros : int
+n_ones : int
+n_zeros_test : int
+n_ones_test : int
+k : int
+alg : String
+read_from_pickle : boolean
+dataset : DataFrame
+dataset_zeros : DataFrame
+dataset_ones : Data frame
+zeros_fraction : float
+ones_fraction : float
+train_zeros : array
+train_zeros_index : array
+dataset_zeros_left : array
+test_zeros : array
+train_ones : array
+train_ones_index : array
+dataset_ones_left : array
+test_ones : array
+dataset_train : array
+dataset_test : array
+selected_feature : array
+X_train : array
+y_train : array
+X_test : array
+y_test : array
+sel : object
+columns : array
+cols_indices : array
+n_estimators : int
+max_features : int
+threshold : float
+simplefilter(String)
+split_data(n_zeros, n_ones, n_zeros_test, n_ones_test, k, alg) : arrays
+read_csv(filepath : String) : DataFrame
+query(String)
+sample(frac : float, optional)
+drop(array) : array
+concat(array) : array
+load(object)
+fit(X_train, y_train) : classifier object
+get_support()
+dump(array, object)
+dataset_info() : String
Neural_Networks
+k : int
+lr : float
+decay : float
+beta_1 : float
+beta_2 : float
+epsilon : float or None
+amsgrad : Boolean
+X_train : array
+X_test : array
+y_train : array
+y_train2 : array
+y_test : array
-features_list : array
+complie(optimizer : String or instance of the optimizer, loss : String or an objective function, metrics : String or a custom metric function) : classifier object
+dataset_info() : String
+evaluate_classification2(model : array, X_test : array, y_test : array, alg : String) : String
+fit(X_train : array, y_train : array, epochs : int, batch_size : int or None, verbose : int) : classifier object
+add()
+dump(String, object)
Powered By Visual Paradigm Community Edition

Figure 3.1: Class Diagram

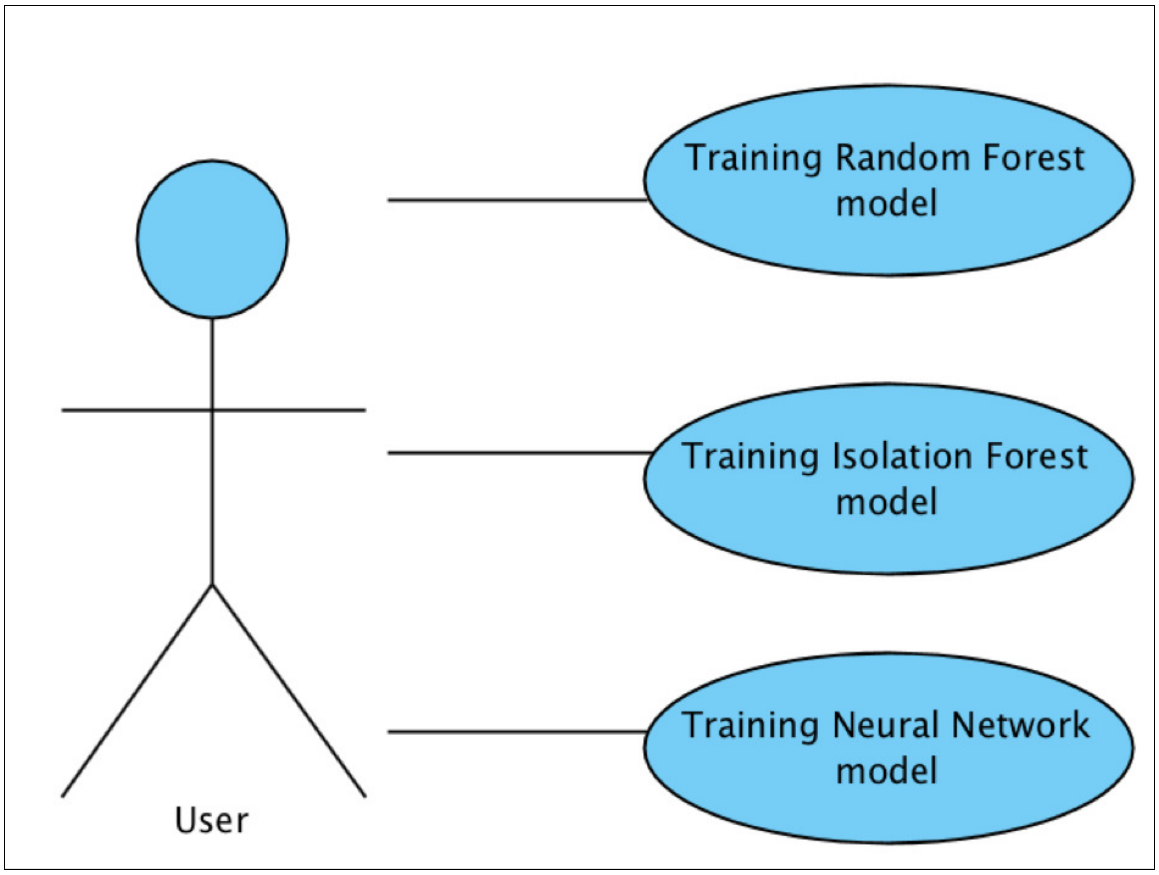

Training Random Forest model
Training Isolation Forest model
Training Neural Network model
User

Figure 3.2: Use Case Diagram

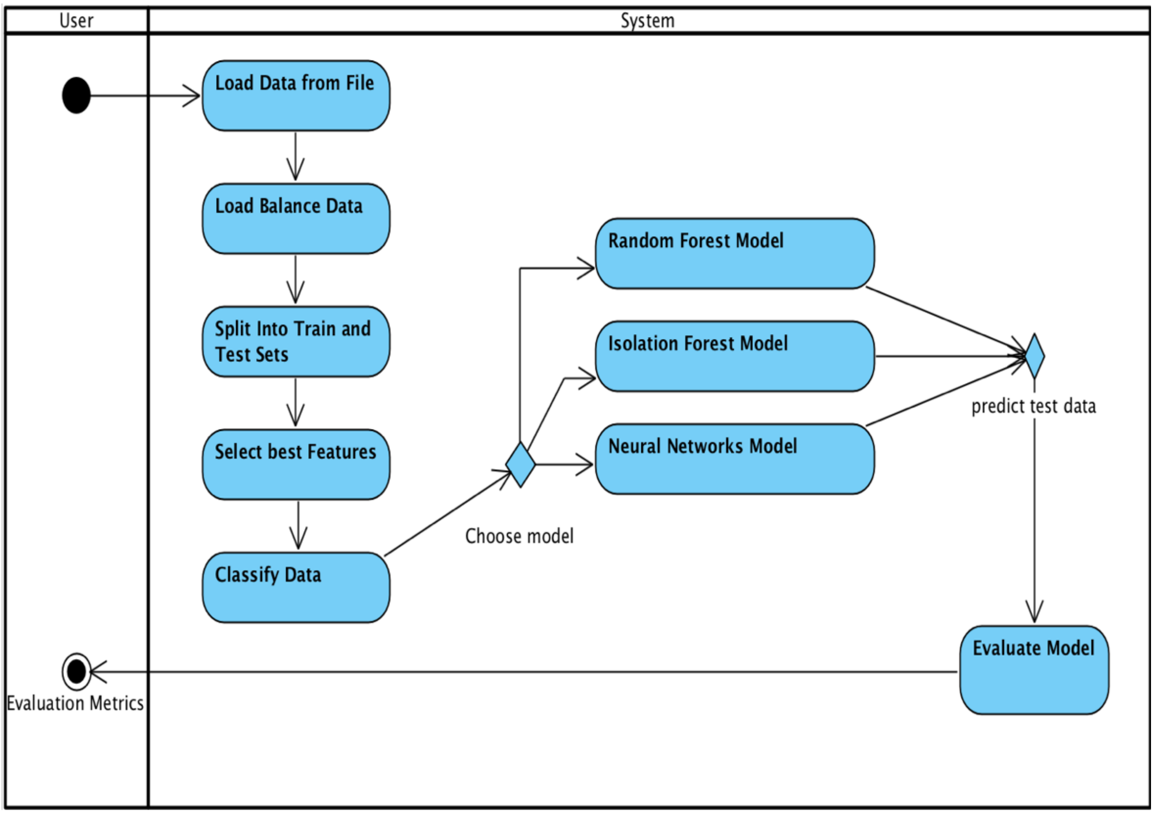


Figure 3.3: Activity Diagram of Software System

A brief explanation of the most important classes will be listed below,

- Evaluation: this class represents the main part of evaluating how good is a classifier model, this will be represented by providing the evaluation performance techniques, which are confusion matrix, accuracy, recall and precision.
- Split_Data: this class is responsible for load and prepare the data set and split it into training and testing sets. By pre-processing the data and creates a DataFrame pandas, it allows loading the data into the system and provides a necessary representation of the data as features and labels to be used later in the software system by the classifiers. It is also important for load balance the fraudulent and legitimate transactions which helps to produce a better model. Also, information about the data sets will be provided including the distribution amount of fraudulent and legitimate transactions used in training and testing.
- Classification_app: This class represents a simple command-line interface to allow the user to provide a data set, and choose a classifier alongside the arguments needed for the data splitting process to create a load balanced training and testing sets, to be used by the chosen classifier. And it represents the final results of the classifier.

Moreover, the use case diagram shown in figure 3.2 represents a grouped set of possible interactions between the system and the user in the software system, where possible classifying options are illustrated. The three available options are Random Forest, Isolation Forest and Neural Networks.
Furthermore, figure 3.3 represents the approach involving dynamic aspects of the implemented software system as an activity diagram. The flowchart involves the steps taken which are; load the data from the given file, load balance the data, split data into training and test sets, select the best features, and finally classify the data using one of the available models. As mentioned previously, the models are Random Forest, Isolation Forest and Neural Networks. Finally, the predicted test set will be resulted and evaluated using the evaluation metrics. Additionally, this section provides all the functional requirements represented in an easy format to read and track.

For better-quality diagrams, soft copies of the diagrams are attached with the submission as external files.

3.4 Development Methodology

In order to tackle the software system that has to be developed and tested in the limited time given, a development methodology had to be followed. The development methodology ensures that developing and testing of the software system are managed, and at the end of the project a functioned prototype will be delivered. Thus, Agile software development strategy was chosen to be followed in this project, as shown in figure 3.2. Agile development strategy is a suitable method, as it focuses on incremental delivery, continual planning and continual learning. It divides the project into iterations, small incremental builds. An advantage of the life cycle model is that the client will be able to see the working outcome straight after each iteration. This methodology was chosen in preference to Waterfall, for example, because if Waterfall was adopted then the implementation stage cannot take place unless the planning stage is done, and due to the fixed structure of the required steps. Which will not be suitable for this project as the timescale is not flexible at all, and the moderation of the requirements belong to this project is at a high risk of changing. However, in Agile any requirements modification at any stage of the process can be integrated even in late development process with no risk in losing the entire work. Moreover, a way faster and successive iteration of a working software system will be produced regularly, and viewed by the client at a consistent pace, which results in client satisfaction with the rapid releases of the project. Lastly, after each iteration the client will be sending feedback that provide a huge opportunity to modify and improve the software in the upcoming iterations. Although Agile is a more teamwork methodology, planning is easier with a single developer. Most of the work came naturally, and I followed it to broke down the tasks that should be done, prioritising the tasks based on feedback given by the supervisor in the meetings, set the intended goals and being flexible to change and modify along the project since the work was partitioned in weekly sprints.

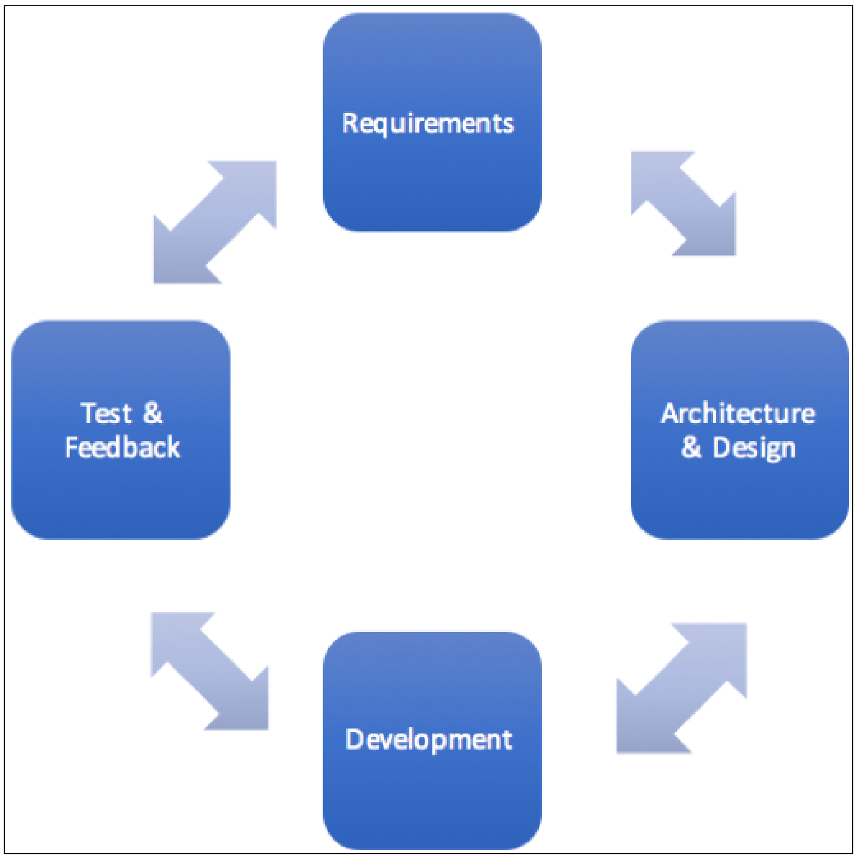


Figure 3.4: Agile Development Strategy

Five iterations have been utilised over the project timescale, objectives and deliverables of each iteration are listed below along with each iteration,

- *First Iteration* focused on implementing, loading and processing the data needed for each classifier.
  - *Deliverable* from this iteration was preparing and splitting the training and testing sets to be ready to use in classification processes.
- *Second Iteration* focused on implementing the three separated classifiers from three different algorithms. Using the data set resulted from the first iteration allow the classifiers to be trained and tested.
  - *Deliverable* from this iteration was producing three classifiers that are able to be trained and tested.

- *Third Iteration* focused on the Load Balancing of both test and train sets.
  - *Deliverable* from this iteration was balance the fraudulent and legitimate transactions to have a well balance model.
- *Fourth Iteration* focused on Feature Selection to train the model.
  - *Deliverable* from this iteration was selecting the features manually or by using the best k features selection method.
- *Fifth Iteration* focused on a simple command-line interface application.
  - *Deliverable* from this iteration was the user interface, in order to provide user interaction with the software, by containing the usage of the three previous iterations.

Chapter 4

Implementation

This chapter will discuss the implementation of the software system provided with the code. The main parts of the implemented code will be specified with the related tools used.

4.1 Project Structure

This project was organised by splitting the code into separated small parts using Python modular programming, to produce a reliable, readable, and maintainable software system. The small parts are represented in classes, that can be collected later to produce a complete software system. Thus, the re-usability of the code is facilitated by the usage of the classifiers, and it ease the access for a specific functionality in the code. Thus, the final solution is implemented and structured in CCFD package that contains six different files demonstrated as follows,

- **Random_Forest:** this classifier file represents a learning model based on Random Forest classification algorithm.
- **Isolation_Forest:** this classifier file represents a learning model based on Isolation Forest classification algorithm.
- **Neural_Networks:** this classifier file represents a learning model based on Neural Networks classification algorithm.

- **Split_Data:** this file is responsible for load balancing, loading the historic data given by the user as an input, prepare them to be classified and split them into fraudulent and legitimate transactions.

- **Evaluation:** this file is responsible for model evaluation as it provides the evaluation techniques metric.

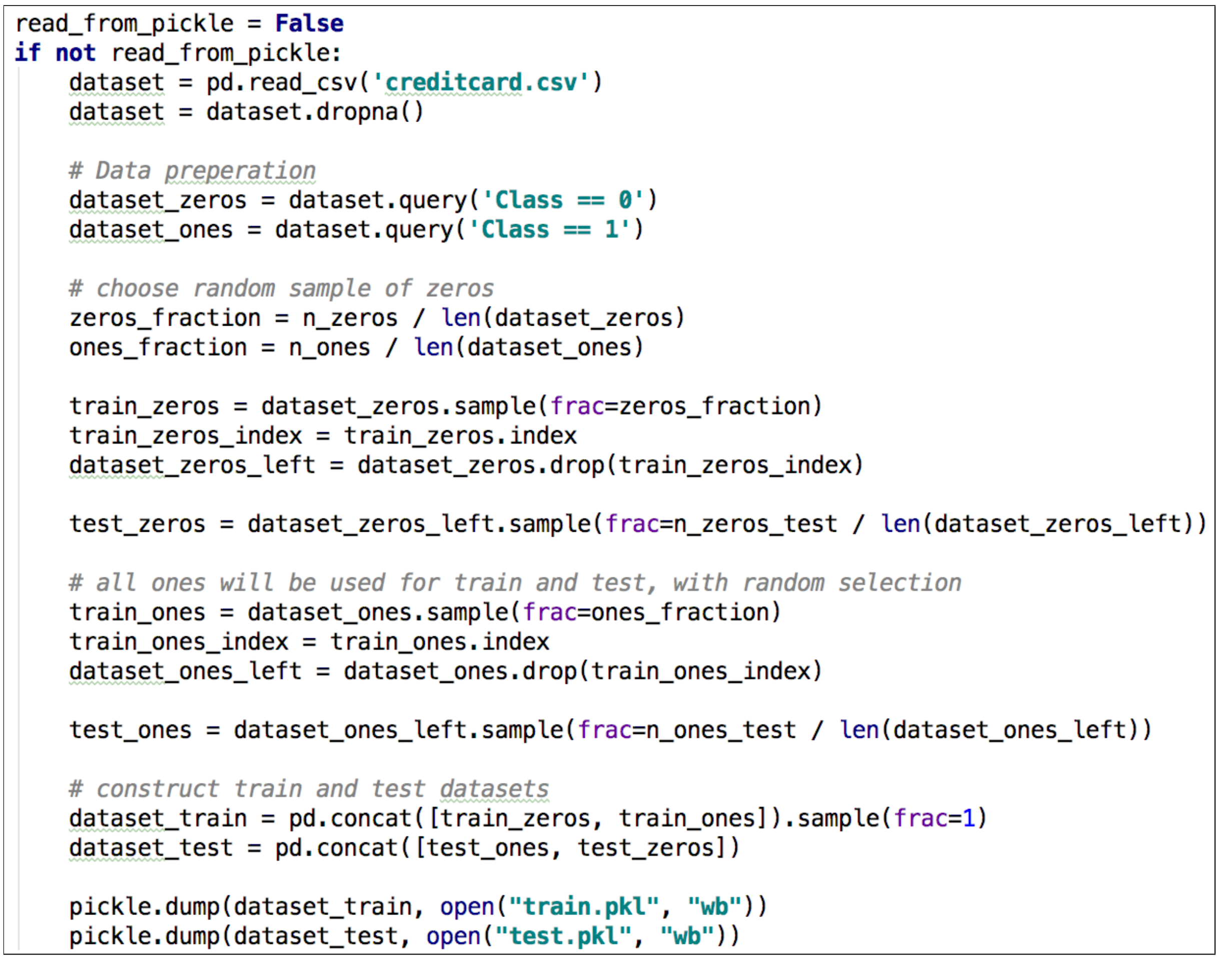
- **Classification_app:** this file represents a simple command line application, that takes the inputs including the directory path of the file and the classifier from the user as arguments, classifies the data, loads and saves the final results of each classifier into models directory, along with the saved output pickled files.

4.2 Data Preparation & Processing

This section will mainly discuss how data set was prepared and processed to be used in each classifier.

4.2.1 Data Loading

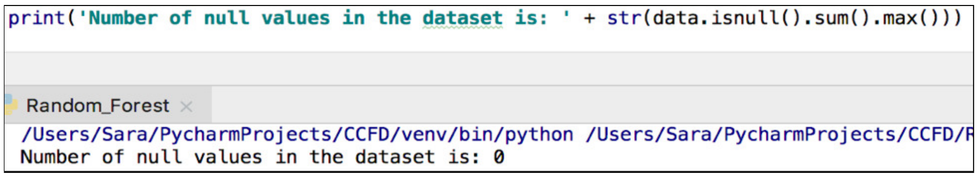
Loading necessary data is the first step in the field of machine learning and its approaches. Since the data consists of historic labelled transactions in csv format, and it must be prepared for the classification algorithms, (built-in function) from csv module, *read_csv()* was imported and called using pandas. The csv file directory will be taken as an argument from the user and will be converted into a DataFrame using pandas for further processing as shown the code block below, figure 4.1.



```
read_from_pickle = False
if not read_from_pickle:
    dataset = pd.read_csv('creditcard.csv')
    dataset = dataset.dropna()

    # Data preperation
    dataset_zeros = dataset.query('Class == 0')
    dataset_ones = dataset.query('Class == 1')

    # choose random sample of zeros
    zeros_fraction = n_zeros / len(dataset_zeros)
    ones_fraction = n_ones / len(dataset_ones)

    train_zeros = dataset_zeros.sample(frac=zeros_fraction)
    train_zeros_index = train_zeros.index
    dataset_zeros_left = dataset_zeros.drop(train_zeros_index)

    test_zeros = dataset_zeros_left.sample(frac=n_zeros_test / len(dataset_zeros_left))

    # all ones will be used for train and test, with random selection
    train_ones = dataset_ones.sample(frac=ones_fraction)
    train_ones_index = train_ones.index
    dataset_ones_left = dataset_ones.drop(train_ones_index)

    test_ones = dataset_ones_left.sample(frac=n_ones_test / len(dataset_ones_left))

    # construct train and test datasets
    dataset_train = pd.concat([train_zeros, train_ones]).sample(frac=1)
    dataset_test = pd.concat([test_ones, test_zeros])

    pickle.dump(dataset_train, open("train.pkl", "wb"))
    pickle.dump(dataset_test, open("test.pkl", "wb"))
```

Figure 4.1: Data preparation and processing

For the purpose of this project, the credit card transactions data set was given, and validating and cleaning processes were performed to make sure that no redundancy, null values or empty spaces in columns included as shown in the code block below, figure 4.2.



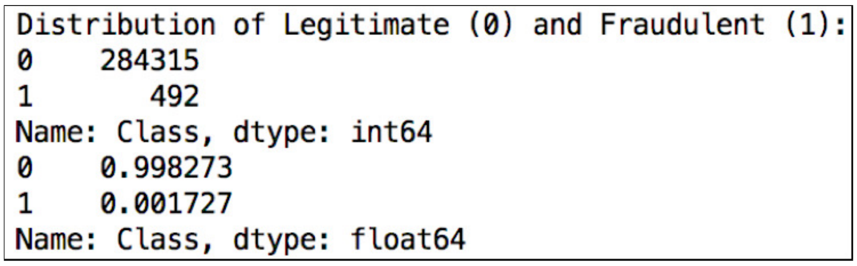
```
print('Number of null values in the dataset is: ' + str(data.isnull().sum().max()))
```

```
/Users/Sara/PycharmProjects/CCFD/venv/bin/python /Users/Sara/PycharmProjects/CCFD/F
Number of null values in the dataset is: 0
```

Figure 4.2: Data validation

4.2.2 Data set Review

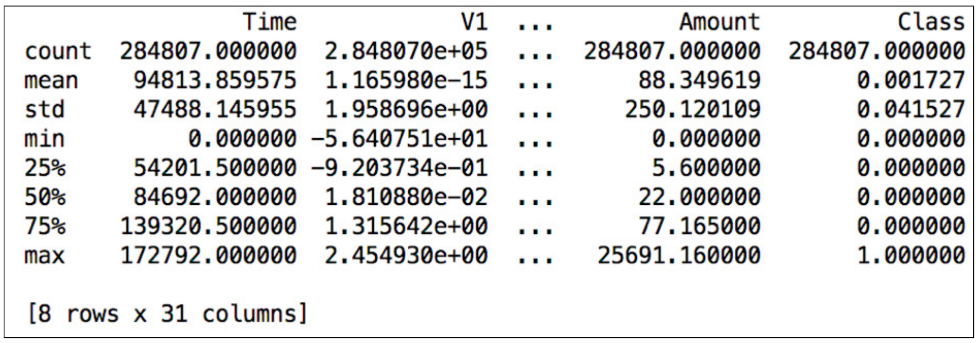
According to [15], the given data set contains credit cards transactions done by European cardholders in 2013. In two days, 284,315 legitimate transactions, and 492 fraudulent transactions occur. It is highly unbalanced data, whereas fraud and legitimate transactions are represented respectively, by approximately 2% and 98% as illustrated in the following code block, 4.3.



```
Distribution of Legitimate (0) and Fraudulent (1):
0    284315
1       492
Name: Class, dtype: int64
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

Figure 4.3: Data distribution

The data set contains only numerical (continues) input variables resulted from a Principal Component Analysis (PCA) feature selection transformation in 28 major components out of the 30 components utilised in this project. The behavioural characteristics of the card is represented by, a variable of each profile usage represents the spending behaviours of the customer linked with hours, and days of the month, as well as geographical locations or the merchant type where the transaction happens. Therefore, to distinguish fraudulent activities these variables are used later. Unfortunately, the details and background information of the provided features cannot be specified due to confidentiality issues. The Time feature involves the seconds passed between each transaction and the first transaction occurrence. The Amount feature represents the amount of the transactions, and it has a relatively small mean of all the transactions made which is 88.3 as in the code shown in figure 4.4 below, and the Class feature is the response variable, as it distinguishes the transactions by labelling them into zeros and ones.



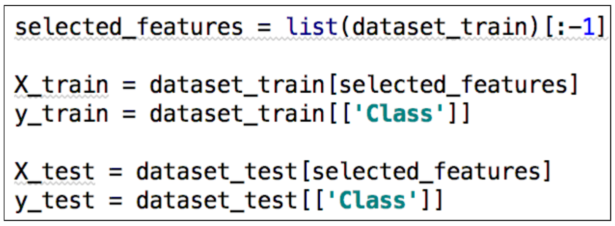
```
                Time            V1  ...         Amount          Class
count  284807.000000  2.848070e+05  ...  284807.000000  284807.000000
mean    94813.859575  1.165980e-15  ...      88.349619       0.001727
std     47488.145955  1.958696e+00  ...     250.120109       0.041527
min         0.000000 -5.640751e+01  ...       0.000000       0.000000
25%     54201.500000 -9.203734e-01  ...       5.600000       0.000000
50%     84692.000000  1.810880e-02  ...      22.000000       0.000000
75%    139320.500000  1.315642e+00  ...      77.165000       0.000000
max    172792.000000  2.454930e+00  ...   25691.160000       1.000000

[8 rows x 31 columns]
```

Figure 4.4: Features descriptive statistics

4.2.3 Train and Test Split

Data set used for classifiers is usually split into training and test data sets. This process has been done to avoid over fitting problem detailed in the background section. Testing the accuracy of a classifier in its prediction of fraudulent transaction using the test set is essential, to know how well the model learned on the training set. Scikit-learn provides useful functions, *fit()* and *predict()*, for train and test the classifiers. Usually, a 70/30 or 80/20 split is used for train and test. However, a load balancing has been conducted in this project instead, and detailed in the next section.

Moreover, the training and testing sets will be constructed at the end as **dataset_train**, and **dataset_test**, to be used and divided into transactions and labels for both training and test sets, as the code in figure 4.5 shown below.

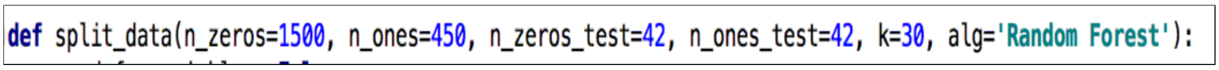


```
selected_features = list(dataset_train)[:-1]

X_train = dataset_train[selected_features]
y_train = dataset_train[['Class']]

X_test = dataset_test[selected_features]
y_test = dataset_test[['Class']]
```

Figure 4.5: Train and test sets

4.2.4 Load Balancing

For this project, it can be seen from the above mentioned information that the data set is highly unbalanced. Attempting to train a model with the given data combination, will result in over fitting as the model will conveniently converge into classifying everything as legitimate. Consequently, the high accuracy of training sets will be resulted as the fraudulent transactions are so few and the outlier fraudulent transactions will never be detected. So, in order to balance the data, the number of zero instances which represent legitimate transactions, and one instances which represents fraudulent transactions used must be balanced. For test sets the amount was fixed to 42 instances for both zeros and ones so that it is could assess how the model classifies true positives and true negatives accurately. In regards to the training sets, the number of one instances was set to 450 as there was no other option, and the number of zeros was adjusted into the data combination chosen throughout different experiments. The number of zero labelled rows was randomly chosen and experiments were run a number of times for each data mixture in order to understand the fluctuation of accuracy due to the varied data samples. A sample of the instance distribution can be seen in figure 4.6 shown below.

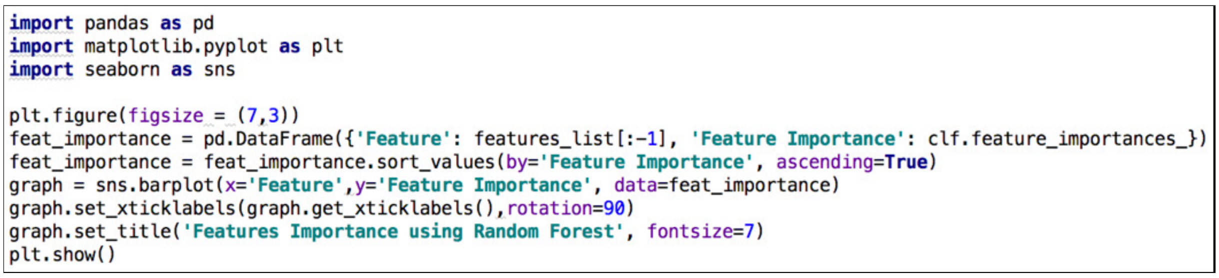


```
def split_data(n_zeros=1500, n_ones=450, n_zeros_test=42, n_ones_test=42, k=30, alg='Random Forest'):
```

Figure 4.6: Instance distribution in *split_data()*

4.2.5 Feature Importance

As mentioned about features selection in the background section, feature importance is a significant quality in ensemble algorithms, specifically in random forests. It is been utilised in this project because it measures the relative importance of each feature in the prediction. The attribute *feature_importances_* was implemented in **Random_Forest** class, it returns an array of numbers of each feature importance determining the splits. Therefore, a combination of Seaborn and Matplotlib libraries were used for a good visualisation of the results in ascending order, the code is shown in figure 4.7 below.



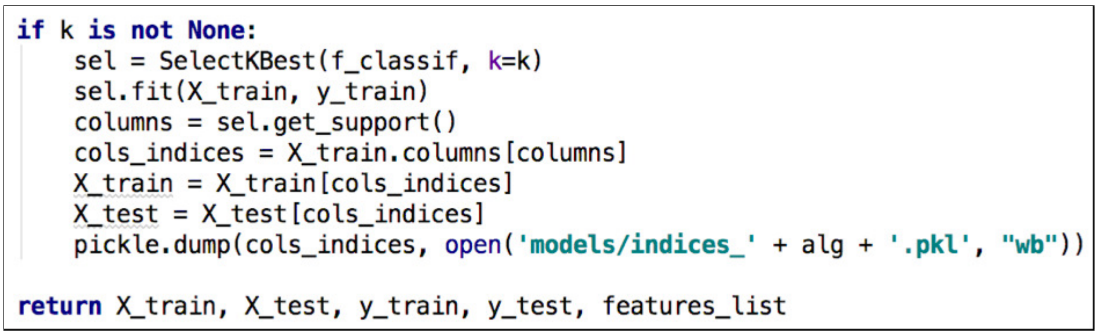
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize = (7,3))
feat_importance = pd.DataFrame({'Feature': features_list[:-1], 'Feature Importance': clf.feature_importances_})
feat_importance = feat_importance.sort_values(by='Feature Importance', ascending=True)
graph = sns.barplot(x='Feature',y='Feature Importance', data=feat_importance)
graph.set_xticklabels(graph.get_xticklabels(),rotation=90)
graph.set_title('Features Importance using Random Forest', fontsize=7)
plt.show()
```

Figure 4.7: *feature_importances_*

4.2.6 Feature Selection

Another important step of data preparation is the feature selection process. This data set provides 32 data headers (features). Experiments were run to choose the k best features of this feature set using *SelectKBest()* from Scikit-learn library.

Choosing features can either be done manually by changing the k value of each classifier in each try, or by setting the k value into None which basically selects all features except the labels, and this option allows the classifier to be trained using the features with the highest scores. The code shown in figure 4.8, also indicates that the method takes two attributes, **f_classif** which represents the ANOVA F-value between a label or feature for the classification tasks, and the k value that represents the features.



```
if k is not None:
    sel = SelectKBest(f_classif, k=k)
    sel.fit(X_train, y_train)
    columns = sel.get_support()
    cols_indices = X_train.columns[columns]
    X_train = X_train[cols_indices]
    X_test = X_test[cols_indices]
    pickle.dump(cols_indices, open('models/indices_' + alg + '.pkl', "wb"))

return X_train, X_test, y_train, y_test, features_list
```

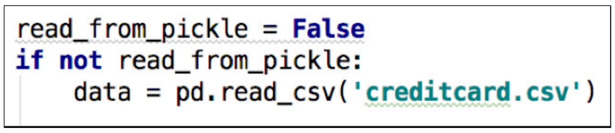
Figure 4.8: *SelectKBest()*

4.2.7 Training and Evaluating the Model

The **selected_features** variable shown in figure 4.5, in train and test split section, will contain the whole list of features except the labels represented by [Class] feature. The labels are assigned by both variables **y_train** and **y_test**. However, This will result in having the training set represented by **X_train** and **y_train**, and test represented by **X_test** and **y_test**. All the variables will be returned by the *split_data()* method. To train the classifier and fit it into the training set the function *fit()* has been implemented in each model. This function requires passing the training set transactions and the labels as arguments. The *predict()* function was also used by the classifiers in the evaluation part to classify the received data points from unseen data, or a testing test in some cases.

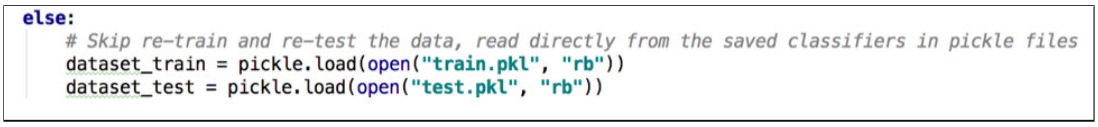
4.2.8 Pre-processed Data

Since random sampling is used in load balance the training data set, a random sample in training set is reproduced in each run of the code. However, for model optimisation the same input data must be used, to avoid producing two different results. Moreover, the observed change in the accuracy is resulted from changing the parameters of the chosen algorithm not from the change of input data. So, pickle is used to save data in order to ensure that the experiments done are reproducible, also to ensure that the input data has not been changed. This will lead to deterministic results, which allows running the same experiment with the same parameters twice and obtain exactly the same results. Since pickle is a persistence model, it was implemented in *split_data()* method. Pickle created object files that will contain classifier models created previously, and it can be saved and loaded. In order to store an object externally, for a classifier object, pickle uses the *pickle.dump()* function to save the data. Whereas in loading the object , *pickle.load()* will be used to de-serialise the data. This allows the user to use a previously saved object of the chosen classifier, if the **read_from_pickle** Boolean variable shown in figure 4.9 is set to True, the code in figure 4.10 will be executed to handle loading the saved classifiers for training and testing.



```
read_from_pickle = False
if not read_from_pickle:
    data = pd.read_csv('creditcard.csv')
```

Figure 4.9: Pickle objects - set the value



```
else:
    # Skip re-train and re-test the data, read directly from the saved classifiers in pickle files
    dataset_train = pickle.load(open("train.pkl", "rb"))
    dataset_test = pickle.load(open("test.pkl", "rb"))
```

Figure 4.10: Pickle objects - loading file

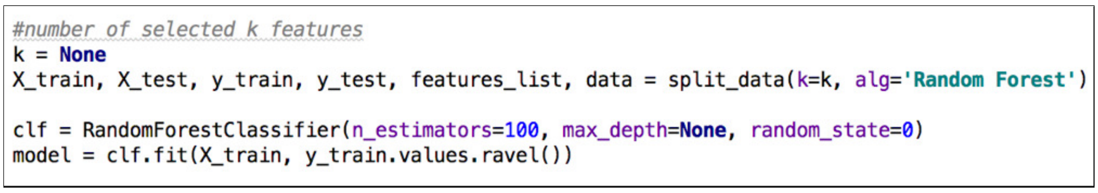
Otherwise, False, will allow the system to create new objects for both training and testing, that can be used in similar further experiments.

4.3 Classifiers

For the purpose of this project, three types of classifiers have been created, and constructed as three separated classes that implement *split_data()* function from **Split_Data.py**, and *evaluate_classification()* function from **Evaluation.py**. Scikit-learn classifiers have been also imported where each classifier class implements the associated model of its learning algorithm. Hence, **RandomForestClassifier** has been constructed from Scikit-learn to represent the model of Random Forest classifier. **IsolationForest** has been also constructed for Isolation Forest classifier, and for Neural Networks **Sequential** model has been imported from Keras library.

4.3.1 Random Forest

Random Forest algorithm is the supervised classifier chosen to predict whether a transaction is legitimate or fraudulent. In order to classify fraudulent transactions by this classifier, RandomForestClassifier has been imported from Scikit-learn ensemble module along with pickle module. As shown in figure 4.11, the hyper-parameters chosen in the implemented code were, the **n_estimators**, representing the tree numbers, and set to 100. Also, **random_state** was set to 0 and the **max_depth** was set to None. Additionally, both **X-train** and **y_train** were used into *fit()* function to create the model.

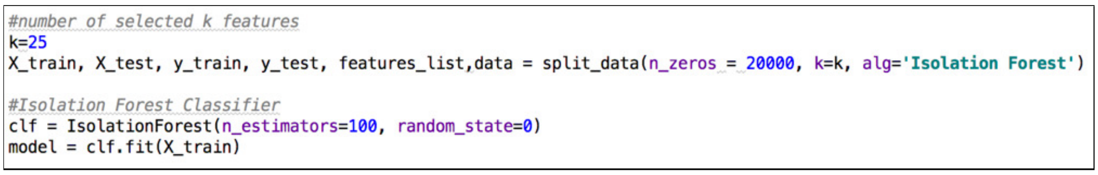


```
#number of selected k features
k = None
X_train, X_test, y_train, y_test, features_list, data = split_data(k=k, alg='Random Forest')

clf = RandomForestClassifier(n_estimators=100, max_depth=None, random_state=0)
model = clf.fit(X_train, y_train.values.ravel())
```

Figure 4.11: Random Forest Classifier

4.3.2 Isolation Forest

Isolation Forest is unsupervised classifier, created to be compared to the supervised and the deep learning algorithm implemented in this project. Along with pickle module, **IsolationForest** was imported from Scikit-learn to create the model by training the data using *fit()* function that takes **X_train** as a parameter. As shown in figure 4.12 below, the hyper parameters chosen in this classifier were **random_state** which was set to zero, and **n_estimators** was set to 100.

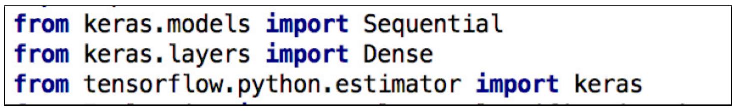


```
#number of selected k features
k=25
X_train, X_test, y_train, y_test, features_list,data = split_data(n_zeros = 20000, k=k, alg='Isolation Forest')

#Isolation Forest Classifier
clf = IsolationForest(n_estimators=100, random_state=0)
model = clf.fit(X_train)
```

Figure 4.12: Isolation Forest Classifier

4.3.3 Neural Networks

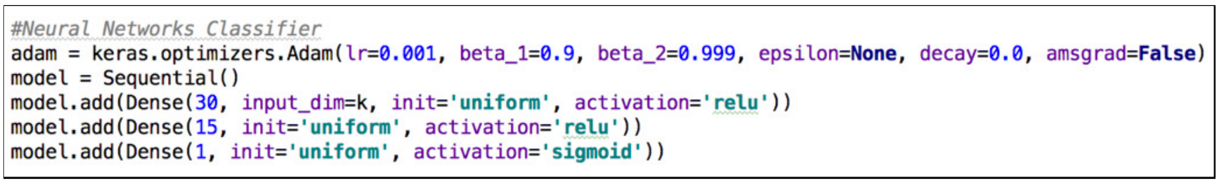
To construct a Neural Network model, Sequential model API alongside a network of dense layers were imported from Keras library. Keras was also imported from Tensorflow as can be seen in figure 4.13.



```
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.python.estimator import keras
```

Figure 4.13: Imported libraries

The first step was building the classifier using Sequential constructor that will have layers added to it using the function *add()* as shown in figure 4.14. The layers added are of type Dense, the 30 represents the output size, whereas 15 represents the output in the second Dense layer, and both can be changed as preference. The option **relu**, Rectified Linear Unit, was chosen as an activation function. Whereas, the last layer presents the predictions using **sigmoid** which is a smoother function to get more accurate probabilities for classification.

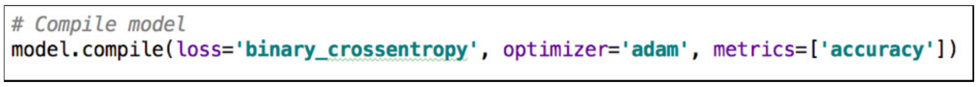


```
#Neural Networks Classifier
adam = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
model = Sequential()
model.add(Dense(30, input_dim=k, init='uniform', activation='relu'))
model.add(Dense(15, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))
```

Figure 4.14: Neural Network Classifier

The k best feature can also be utilised using neural networks if the features used in the k value are set to all, using the value None, that allows passing all features for the *SelectKBest()*.

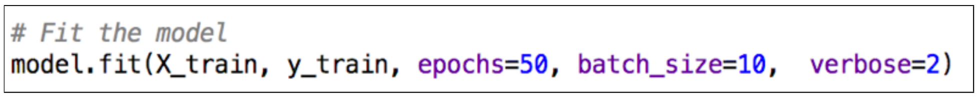
Compiling the model and the learning process configuration is the next step in neural networks classifier. The method *compile()* of the Sequential model was used, and it requires three arguments to be passed, a loss function, an optimizer which set to **Adam** and a list of metrics. The three passed arguments are presented in figure 4.15 below.



```
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figure 4.15: Compiling the model

Last step was fitting the model using *fit()* function which requires at least two arguments: input and target tensors, noting that only a single iteration of the training data will be performed when passing two arguments. In the case of this project, it is shown below that five arguments have been passed, **X_test** and **y_test** represnting the training set, **epochs** which represents the number of batches, **batch_size** which is the number of the samples passed, and **verbose**which is the output logging as shown in figure 4.16.



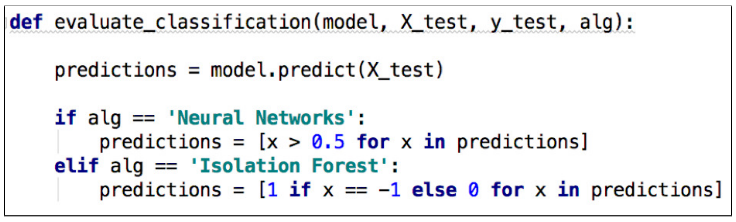
```
# Fit the model
model.fit(X_train, y_train, epochs=50, batch_size=10,  verbose=2)
```

Figure 4.16: Fitting the model

The evaluation performance was calculated and printed out using the *evaluate_classification()* function as what have been done with the other classifiers.

4.3.4 Classifiers Evaluation

In order to evaluate the implemented classifiers, the evaluation method *evaluate_classification()* function was imported from **Evaluation.py**. Scikit-learn Confusion Matrix, Accuracy, Precision and Recall classification metric functions have been imported in this class, and utilised as specified in the learning evaluation section in the background. Each evaluation function requires passing the classification algorithm along with the testing sets and the predictions. However, the prediction is calculated differently for each algorithm as illustrated in figure 4.17 shown below. For random forest, for example, it is declared to be used as the default, whereas in neural networks and isolation forests, -1 will be returned for outliers (Fraudulent), and 1 (Legitimate) for inlier outputs. Therefore, this alteration was done to ensure that classification outputs are consistent across all three algorithms and that the system will automatically trigger the chosen one. Prediction along with other necessary variables are used in the method. This method calculates and prints out the confusion matrix, and construct the accuracy, recall, and precision scores, that will be listed for each classifier results later on the Evaluation section.



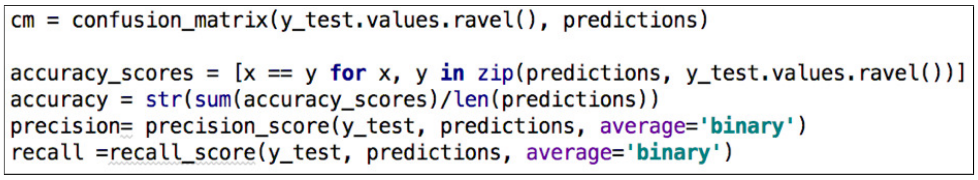
```python
def evaluate_classification(model, X_test, y_test, alg):

    predictions = model.predict(X_test)

    if alg == 'Neural Networks':
        predictions = [x > 0.5 for x in predictions]
    elif alg == 'Isolation Forest':
        predictions = [1 if x == -1 else 0 for x in predictions]
```

Figure 4.17: *evaluate_classification()*

Each evaluation metric with the function used from Scikit-learn, is listed below,

- Confusion Matrix: *confusion_matrix()*
- Accuracy: *accuracy_score()*
- Precision: *precision_score()*
- Recall: *recall_score()*

Figure 4.18 below, indicates the implementation part of the evaluation,

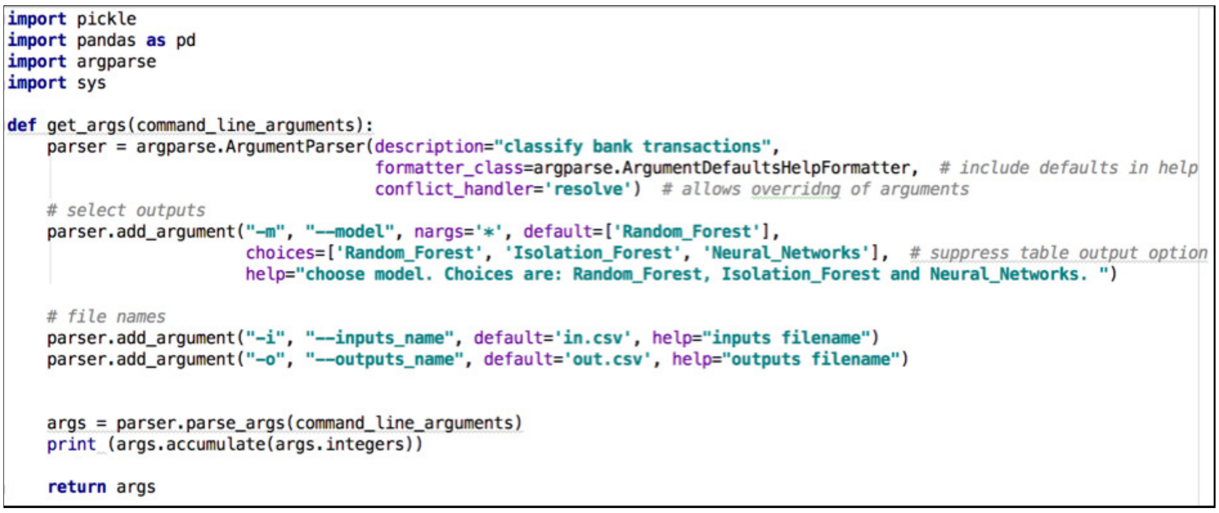


```python
cm = confusion_matrix(y_test.values.ravel(), predictions)

accuracy_scores = [x == y for x, y in zip(predictions, y_test.values.ravel())]
accuracy = str(sum(accuracy_scores)/len(predictions))
precision= precision_score(y_test, predictions, average='binary')
recall =recall_score(y_test, predictions, average='binary')
```

Figure 4.18: Evaluation metric scores

4.4 Classifiers Command-line Interface

A simple command-line interface classification application was utilised, as discussed in the specification and design part previously. It has been utilised for number of reasons. Firstly, it is user-friendly and easy to use, it provides different classification algorithms can be chosen by the user depending on the entered argument, it automatically generates help and usage messages as well, and provide issues errors when the user input invalid arguments. To support this solution, **argparse** python module was utilised, and the simple command line application has been implemented in **classification_app.py**. The *get_args()* function has **ArgumentParser** which is an object that holds all the needed information to parse a command line into python data types, and it has a list of arguments with different options available added using *add_argument()* function, figure 4.19 below shows the implemented code of this part.



```
import pickle
import pandas as pd
import argparse
import sys

def get_args(command_line_arguments):
    parser = argparse.ArgumentParser(description="classify bank transactions",
                                     formatter_class=argparse.ArgumentDefaultsHelpFormatter,  # include defaults in help
                                     conflict_handler='resolve')  # allows overridng of arguments
    # select outputs
    parser.add_argument("-m", "--model", nargs='*', default=['Random_Forest'],
                        choices=['Random_Forest', 'Isolation_Forest', 'Neural_Networks'],  # suppress table output option
                        help="choose model. Choices are: Random_Forest, Isolation_Forest and Neural_Networks. ")

    # file names
    parser.add_argument("-i", "--inputs_name", default='in.csv', help="inputs filename")
    parser.add_argument("-o", "--outputs_name", default='out.csv', help="outputs filename")


    args = parser.parse_args(command_line_arguments)
    print (args.accumulate(args.integers))

    return args
```

Figure 4.19: Simple command-line application

Moreover, the information is saved and used by *parse_args()* function. The arguments will be added up using the *accumulate()* function.

In order to run the application, the user must pass two arguments to the command line, the model of choice (ie. Random forest, Neural networks or Isolation forests) and the path of the file containing the data set to classify. Once the arguments are passed, the app chooses the corresponding serialized classification model as well as the selected feature indexes for it, which have been saved as serialized pickled objects after training. For this reason it is important that any new data set, must follow the same structure as the training data set given.

Chapter 5

System Evaluation

This chapter involves test cases that were done to guarantee and demonstrate that the software system produced satisfies the requirements and it works as intended. All the test cases provided in the Software System Testing and Evaluating section below, have been carried out on Mac operation system device version 10.13.6, using Terminal to run the software.

5.1 Software System Testing and Evaluating

The test cases provided below, carried out on the three most significant created functions in the system.

Test Case: 1	Test Purpose: Classification app works with random forests		
Prerequisites: A trained Random Forest model and a set of feature indices			
Test Case Steps: 1. Run the app with an unknown data set. 2. Collect classification results from output folder.			
Step No	Procedure	Response	Pass/Fail
1	Run app with x and y input as command line arguments.	Exit with code 0 (No errors)	Pass
2	Inspect outputs folder	Output file is there	Pass
Comments: This test case passed all steps.			
Related Tests: None			

Table 5.1: Test Case 1

Test Case: 2	Test Purpose: Classification app works with Isolation Forest		
Prerequisites: A trained Isolation Forest model and a set of feature indices			
Test Case Steps: 1. Run the app with an unknown data set. 2. Collect classification results from output folder.			
Step No	Procedure	Response	Pass/Fail
1	Run app with x and y input as command line arguments.	Exit with code 0 (No errors)	Pass
2	Inspect outputs folder	Output file is there	Pass
Comments: This test case passed all steps.			
Related Tests: None			

Table 5.2: Test Case 2

Test Case: 3	Test Purpose: Classification app works with Neural Networks		
Prerequisites: A trained Neural Networks model and a set of feature indices			
Test Case Steps: 1. Run the app with an unknown data set. 2. Collect classification results from output folder.			
Step No	Procedure	Response	Pass/Fail
1	Run app with x and y input as command line arguments.	Exit with code 0 (No errors)	Pass
2	Inspect outputs folder	Output file is there	Pass
Comments: This test case passed all steps.			
Related Tests: None			

Table 5.3: Test Case 3

Out of the test cases tables shown above, none has failed, and all test cases passed. This means that all of the requirements were met and satisfied.

Consequently, this demonstrates that the approaches acquired in solving the main problem of the project and the implementation of the software system done have been a successful choice as it resulted in carrying a successful software system.

Chapter 6

Results and Discussion

Evaluating the classifiers implemented in the software system, and analysing the performance of each classifier became possible after testing the software system implementation, and finding out that all the test cases are passed and met the requirements successfully.

6.1 Classifier Performance

Based on classification, this report focused on implementing three types of classifiers, Random Forest, Isolation Forest and Neural Networks. And the three classifiers performance were evaluated using the performance evaluation techniques individually.

6.1.1 Random Forest Performance

The figure 6.1 shown below, demonstrates the average accuracy rate of the data set features, resulted by adding one feature a time, using the random forest classifier. The best accuracy rate obtained was a perfect 1.0, resulted of adding up the remaining 30 features, with a training set that includes 800 zero instances, and 450 one instances. Whereas, 42 instances of both zeros and ones were set to balance the test set variables.

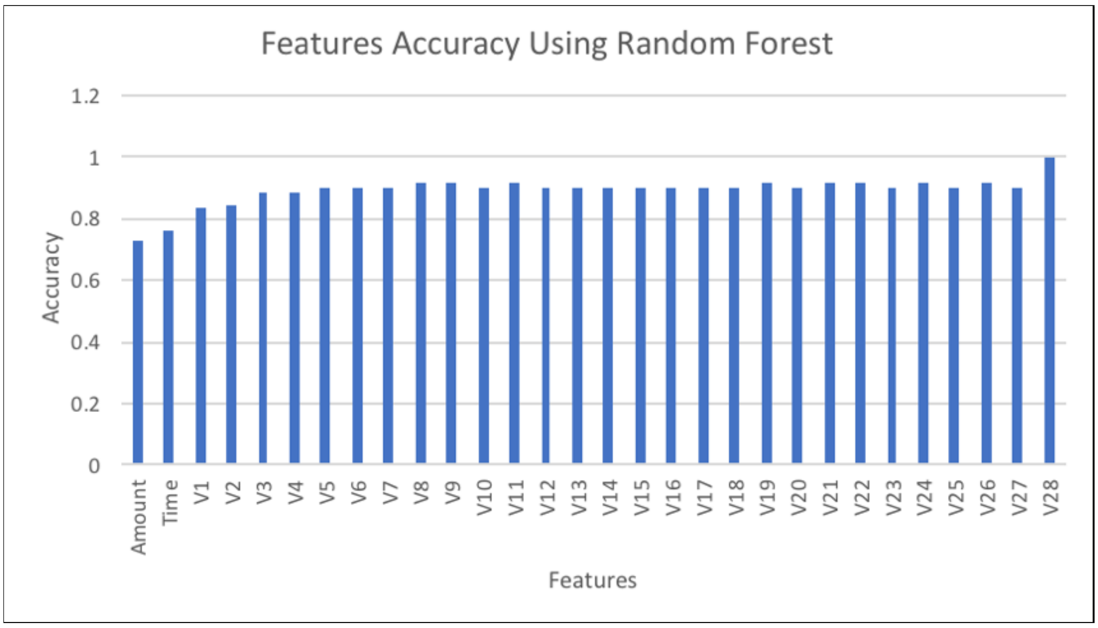


Figure 6.1: Accuracy score of features using RF classifier

Feature Selection

A number of experiments have been done to ensure the best accuracy can be achieved by Random Forest classifier. The features and observations are not seen by all the trees, which assures the de-correlation of the trees consisting the random forest, which are four to twelve hundred decision trees, therefore less prone to over fitting. In these experiments an exploration of feature selection using Scikit-learns *SelectKBest()* features was observed. The training set for these trials contained a sample of 1500 zero instances and 450 one instances, and the test set has 42 instances of both zeros and ones, to compare different number of features, 5, 10, 15, 20 and All which allows all features to be included. The reason that the experiment was done by the chosen instances over the 800 zero samples of the previous section, was to avoid the perfect 100% score obtained. The impact of selecting the k best features would be more visible this way, as if the score obtained was hitting 100% when comparing k best and all, it would be hard to argue which performs best.

Figures 6.2, 6.3 and 6.4 have been created to illustrate the performance evaluation scores of accuracy, precision and recall that were obtained from testing the Random Forest classifier in five different tries.

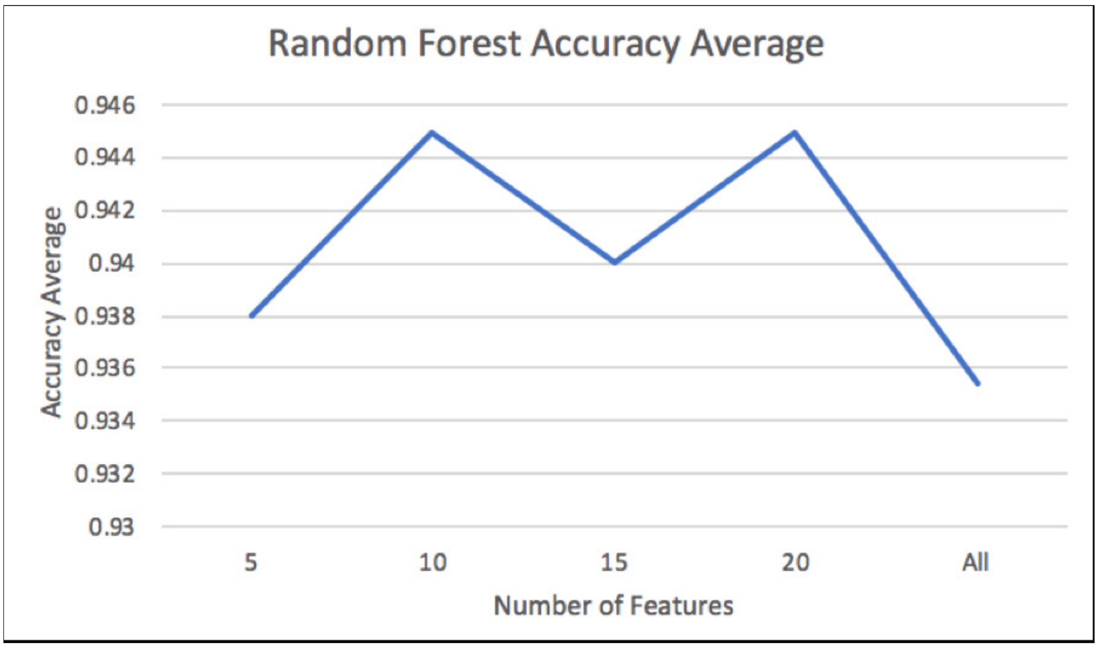


Figure 6.2: Average accuracy of RF classifier with different amount of features

It can be clearly shown in figure 6.2 that there are two peaks in the accuracy performance achieving a score of 0.95, using the selection of 10 and 20 features that were used for training and testing the model. Choosing one of the selections relies on whether the classifier performance is better in precision or recall evaluation. In this case, it can be seen that the best score achieved was with the 20 feature selection as it gives slightly better precision and recall scores than the scores achieved with 10 selection.

However, as shown in figure 6.3 below, the precision average remained constant from 5 to 15 features, and it increased rapidly when 20 features were selected as it reached a perfect 1.0 score. Whereas, recall average demonstrated in figure 6.4 varies among the feature selections, but it was at its highest when 10 features were selected as it resulted in approximately 0.90 score.

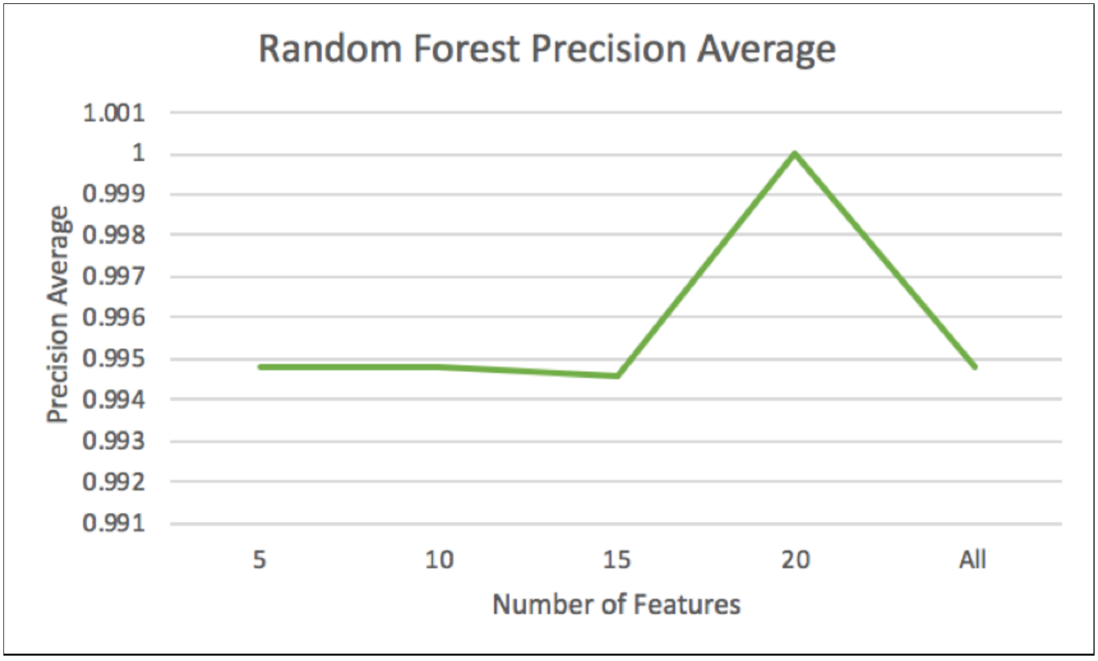


Figure 6.3: Average precision of RF classifier with different amount of features

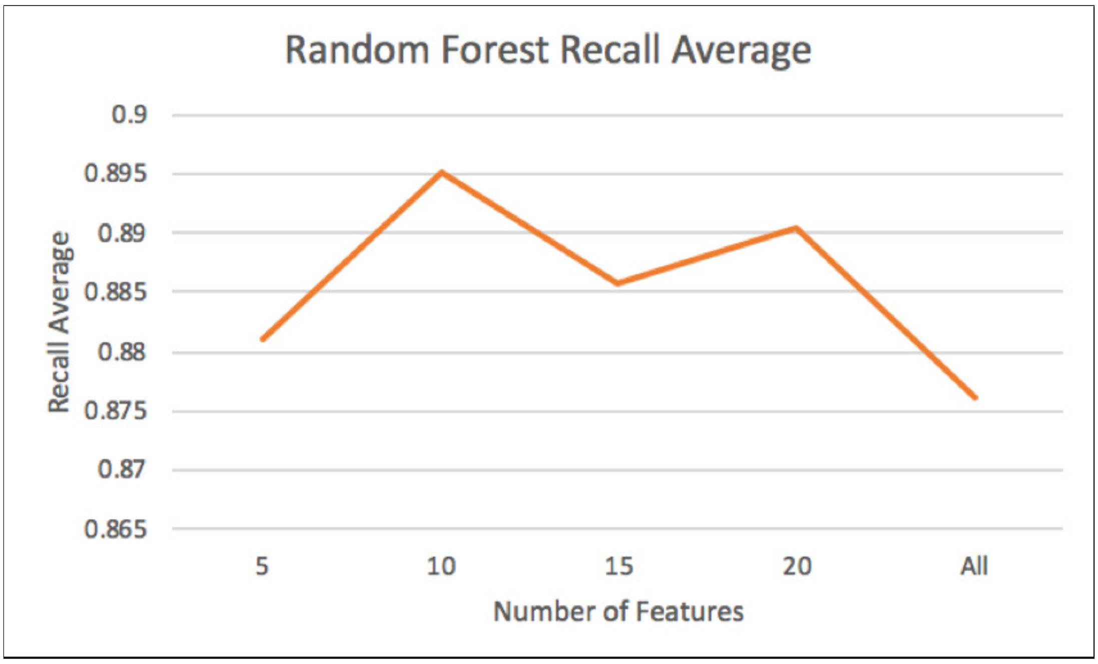


Figure 6.4: Average recall of RF classifier with different amount of features

Feature importance

Feature importance was a part of examining the features available in the data set using random forest classifier, as this tool provides a good indicator of the feature importance among other classifiers used. The results shown in figure 6.5 below, were obtained in an ascending order, where it highlights that the most three important features are V14, V10, and V17.

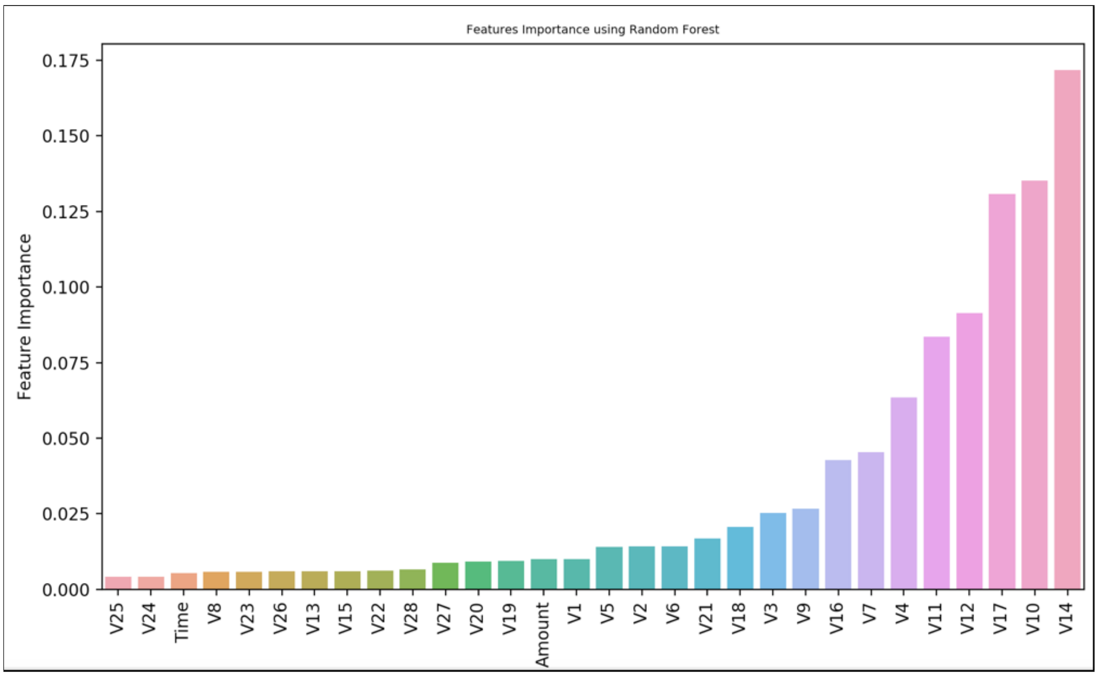


Figure 6.5: Feature Importance using RF classifier

Besides the most important three features, there were some others that can be combined to avoid the over fitting problem, such as, V12, V11, V4, V7, V16 and V9. An experiment was conducted on random forest in 5 tries with the above mentioned features, and it resulted in 0.93 average accuracy score, whereas the precision and recall scored on an average of 0.99 and approximately 0.87 respectively.

Load Balancing

To maintain load balancing in training set, the Random Forest model has been trained in 9 different tries, and different testing results were obtained from the balanced test set that has 84 instances, 42 instances of each zeros and ones. However, as illustrated in figure 6.6 below, the model was firstly trained using all the features, since the best accuracy result was given when all features were selected, as previously mentioned and shown in figure 6.1 above, but with different amount of the zeros instances that represents the legitimate transactions in the training set, and with a fixed 450 instances of ones left out from the total.

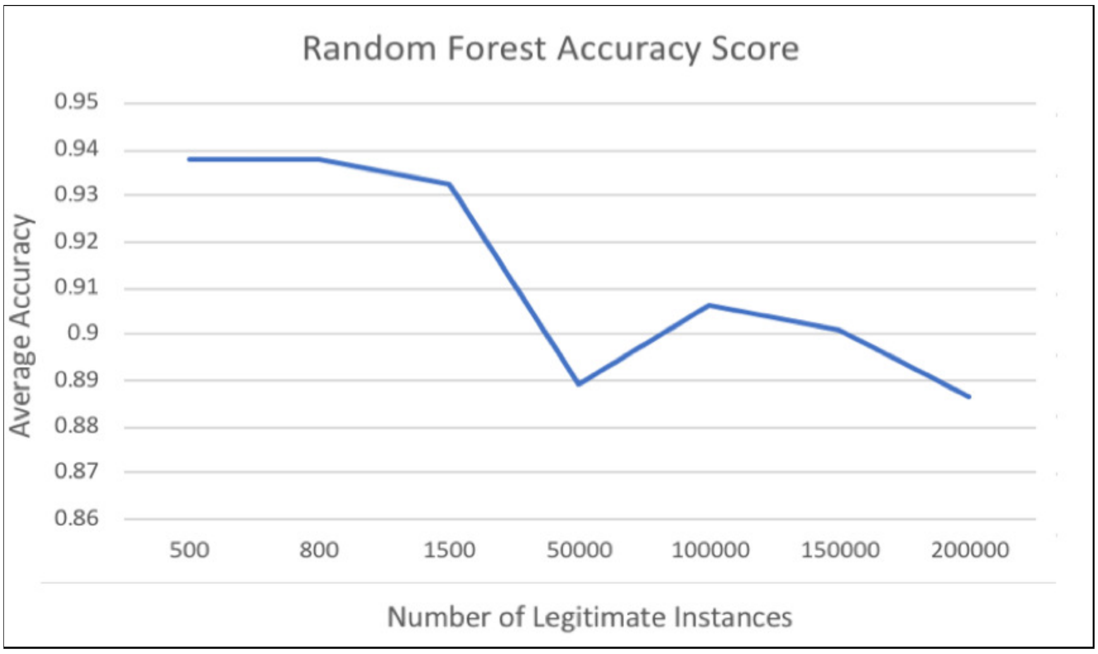


Figure 6.6: RF classifier accuracy with different amount of legitimate instances, using all features

Whereas in figure 6.7, the model was trained using the same training and testing set, but with all features represented by the k value using *SelectKBest()* function. These results were produced to check which feature selection can result in better accuracy scores. It is noticeable that in random forest classifier, with the given unbalanced data set, which is the case in this project, the results were better when the training set has more balanced zeros and ones, as the accuracy score reached almost 0.96 when the amount of zero instances added were ranged between 500 and 800 as illustrated in figure 6.7.

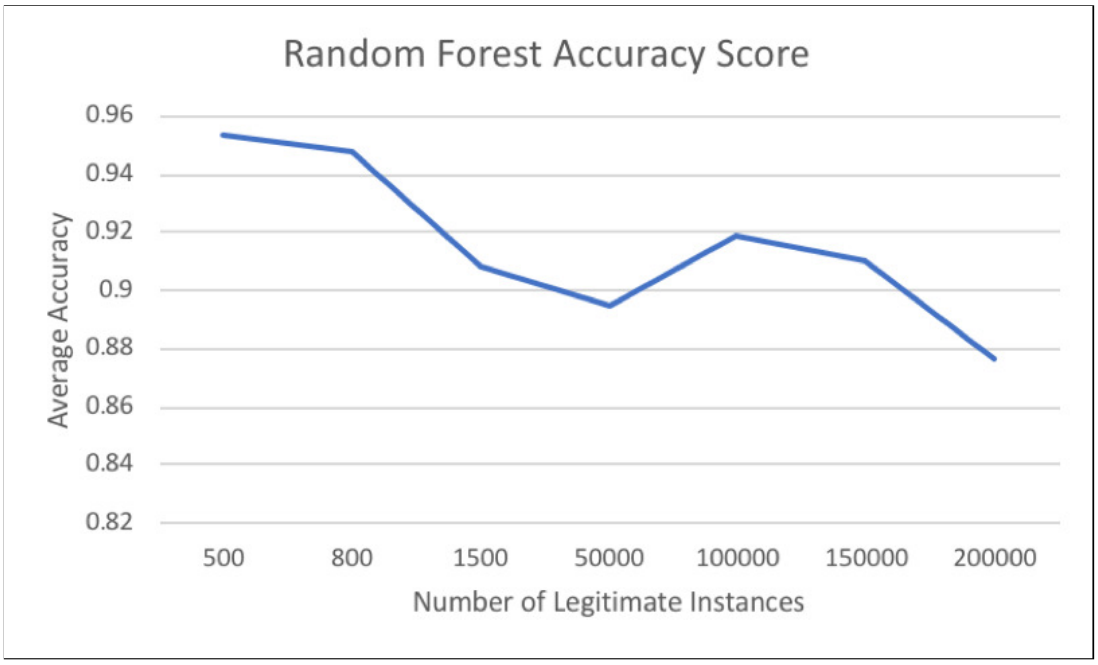


Figure 6.7: RF classifier accuracy with different amount of legitimate instances, using the highest score features

Also, in both figures 6.6 and 6.7, it is clearly shown that better results were obtained roughly between the range of 500 and 800 instances of (zeros) legitimate transactions. Additionally, the accuracy score significantly dropped when the **n_zeros** was set to 50,000 instances. Another observation from both graphs is that the accuracy score dramatically increases between the 50000 to 100000 instances, followed by a new decline with higher values. This indicates that the training set of both instances were not balanced (over fitting) in the training model, and that affect the testing predictions of the model. Therefore, the default was set to 800 zero instances and 450 one instances, and a well-balanced test set has been also created, and divided equally to 42 instances for both zeros and ones.

6.1.2 Isolation Forest Performance

Similar to Random forest performance evaluation, the figure 6.8 shown below demonstrates the isolation forest classifier average accuracy rate of the data set features, represented by adding one feature a time in three tries. The best accuracy rate obtained was 0.94 resulted of adding up 22 features, with a training set that includes 200000 legitimate instances, and 450 fraudulent instances.

In contrast to the Random Forest test, this algorithm does not reach a perfect 1 in the classification evaluation.

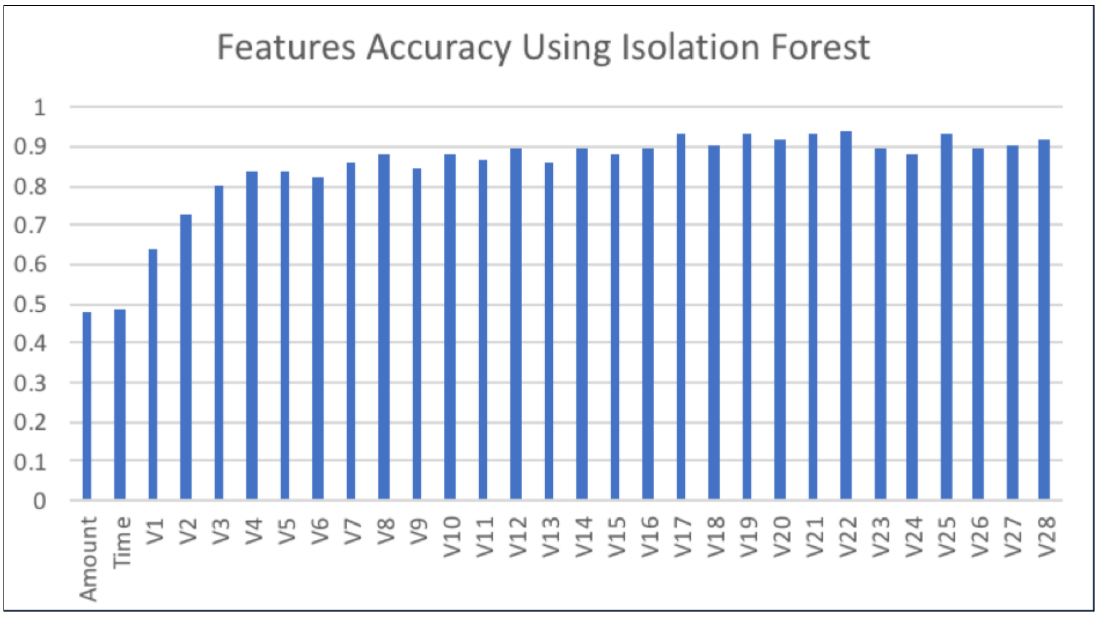


Figure 6.8: Accuracy score of features using IF classifier

The instances were larger in isolation forest because after several test with different instances, this algorithm seems to work better with a huge data set and several dimensions, which referrers to different features available in the data set, where outliers and inliers are included.

The test set was distributed in the same way that was utilised in random forest. Moreover, the features can be selected similarly to random forest, either manually or by selecting all features using the *SelectKBest()* function. Also, feature selection and load balancing were considered in isolation forest approach as well to evaluate the classifier in different experiments.

Feature Selection

Looking into isolation forest classifier, a similar approach to random forest classifier has been taken to evaluate the performance techniques of the model. Accuracy, precision and recall of the classifier have been tested with different k values, range between 5 and 20, as well as testing it with All features.

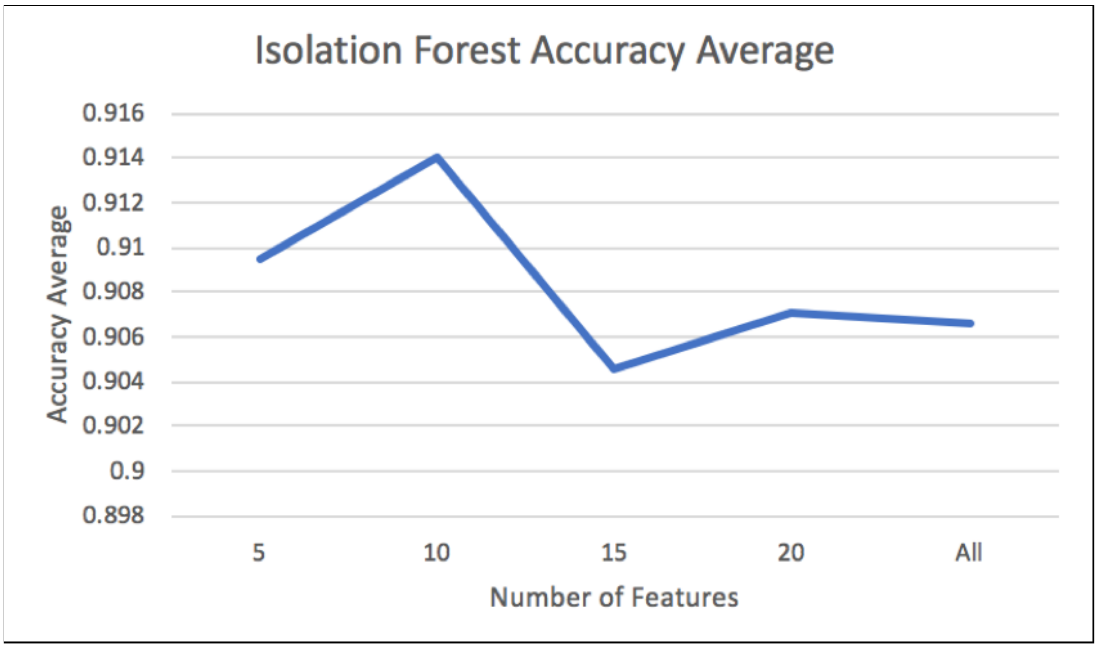


Figure 6.9: Average accuracy of IF classifier with different amount of features

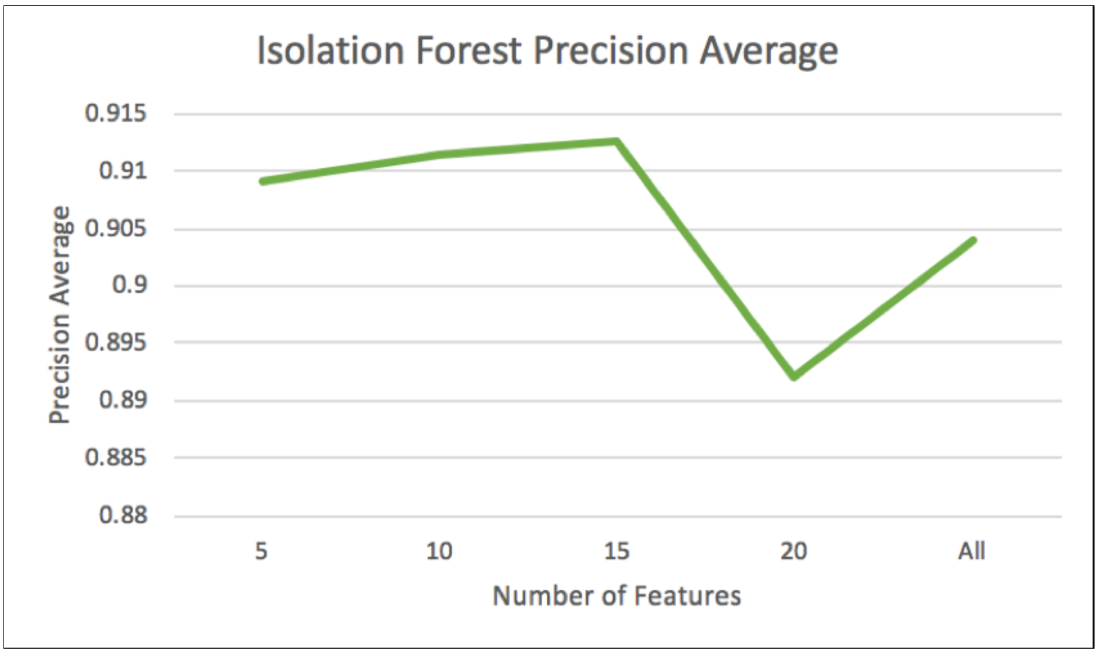


Figure 6.10: Average precision of IF classifier with different amount of features

The figure 6.9 shown above, indicates that training and testing the classifier maintained a high accuracy score between 0.90 and 0.91, whereas the highest score achieved was 0.914 by selecting 10 features and that means the algorithm performs better with this selection, as the precision and recall scores were also high and above 0.91.

Furthermore, precision score was sustained between 0.89 and almost 0.92 among all selections as shown in 6.10. Yet the lowest precision score obtained was by selecting 20 features, whereas the same selection scores the highest recall the highest score achieved 0.93.

Also, as shown in figure 6.11 below, the recall scores achieved by the classifier maintained 0.89 and 0.93 as the highest in the same figure, with different feature selections, so it outperforms the random forest classifier in predicting positive instances correctly.

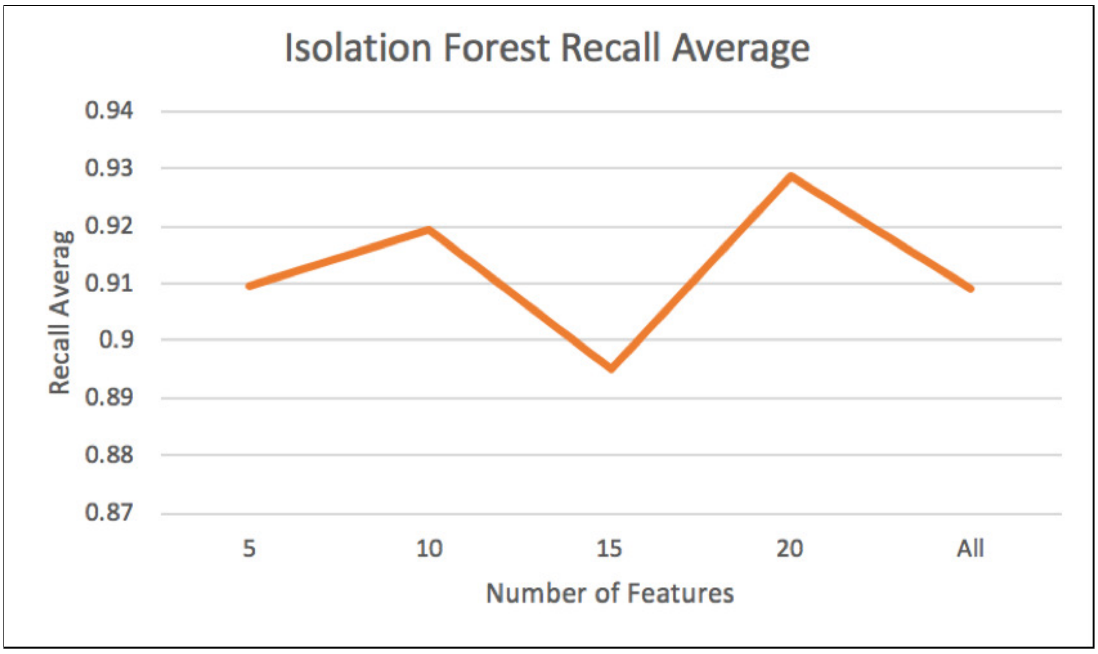


Figure 6.11: Average recall of IF classifier with different amount of features

Load Balancing

As illustrated in figure 6.12 below, while the highest weight features were selected in the model using *SelectKBest()* function, the highest achieved accuracy score of isolation forest classifier was 0.90. The accuracy score increased dramatically from approximately 0.60 up to 0.90, whilst the zero instances in training and test sets are increasing, with a fixed amount of 450 one instances.

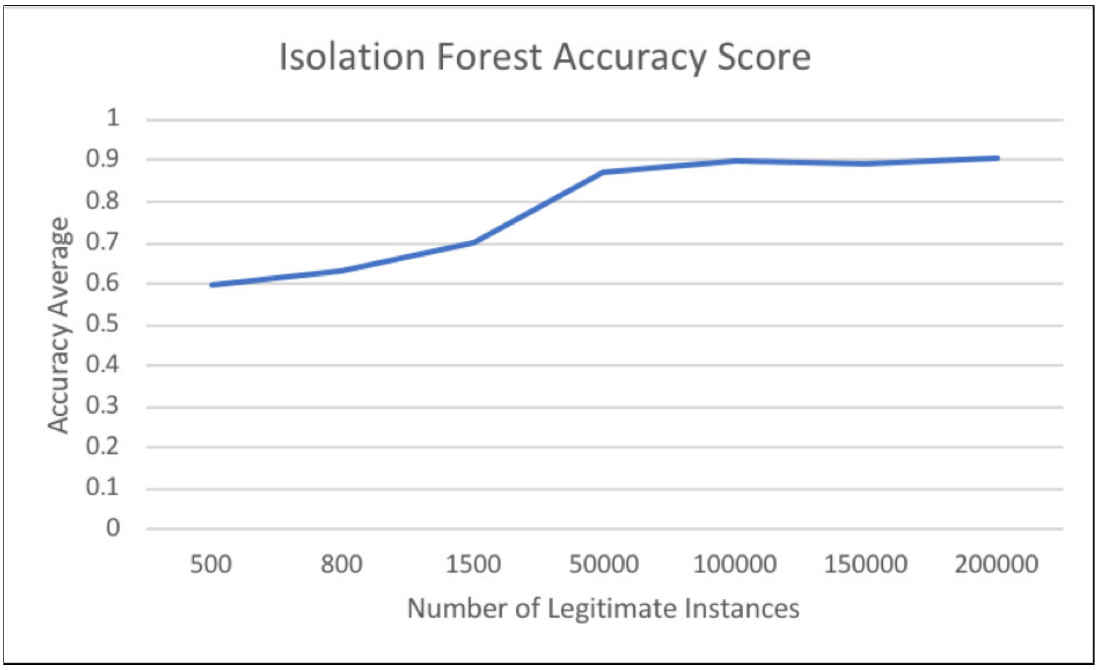


Figure 6.12: IF classifier accuracy with different amount of legitimate instances, using the highest score features

However, with the same number of the one instances, and with the k value set to include all features in the data set manually, into the training and testing sets, figure 6.13 indicates that the results of these experiments were almost in the same range, between 0.60 and 0.90. The highest accuracy score achived was approximately 0.91. Also, the accuracy results became stable beyond 100,000 zero instances in the case of this data set.

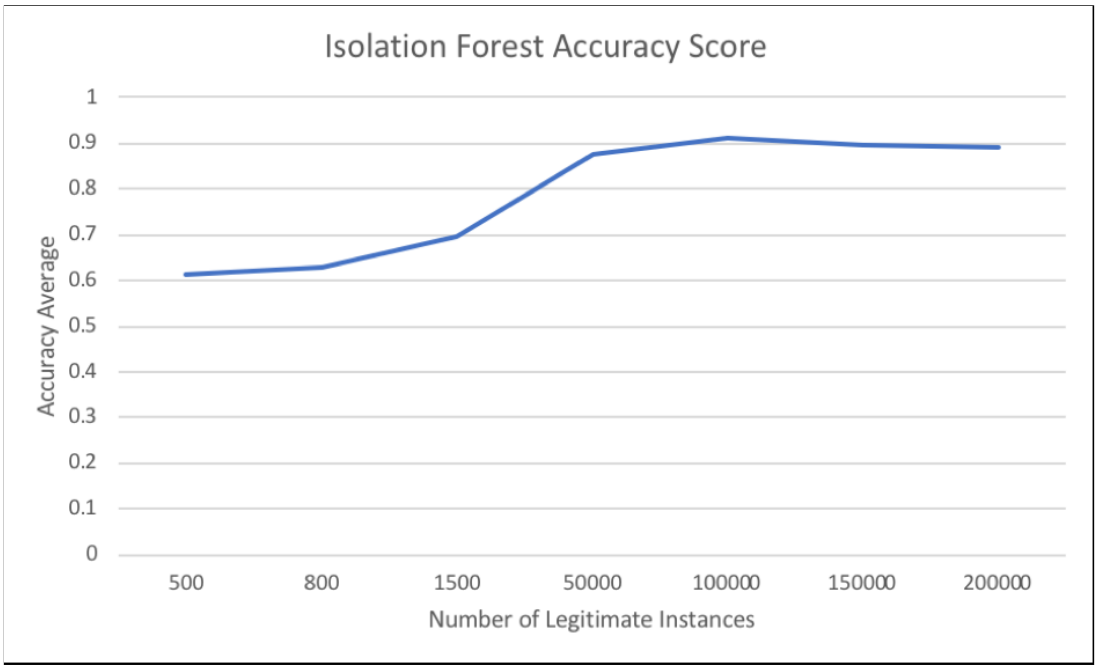


Figure 6.13: IF classifier accuracy with different amount of legitimate instances, using all features

It is evident from the above shown graphs, that unlike Random Forest classifier, the isolation forest algorithm does not favour load balancing, but rather it prefers an unbalanced sample. It makes sense as it is an outlier detection classification algorithm, that works differently to the conventional machine learning ones.

6.1.3 Neural Networks Performance

Similar to the above mentioned performance evaluation of both classifiers, Random Forest and Isolation Forest, the figure 6.14 shown below, demonstrates the neural networks classifier average accuracy rate of the data set features by adding one feature a time in three tries. The best accuracy rate achieved was 0.99. This was a result of adding up only 4 features,and a training set that includes 500 zero instances, and 450 one instances. Similar to the other classifiers, various experiments have been conducted on neural networks, and this was the best load balancing the algorithm seems to perform better with, among other combinations.

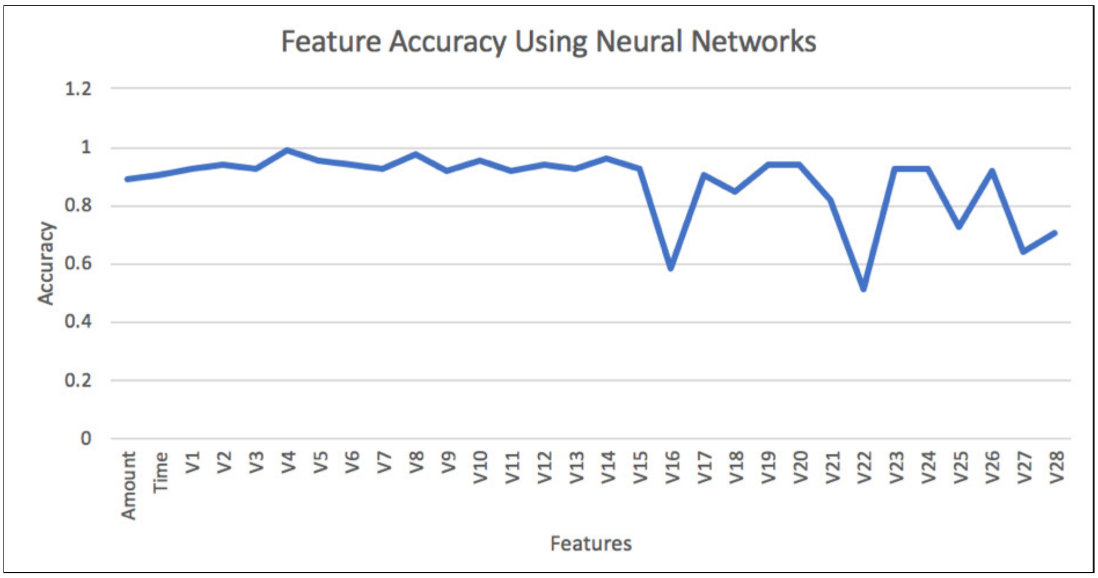


Figure 6.14: Accuracy score of features using NN classifier

In contrast to the Random Forest tests, this algorithm does not reach a perfect 1 in the classification evaluation. Also, unlike Isolation Forest, neural networks seems to work better with a load balanced data sets. However, the test set was distributed in the same way that was utilised in both random forest and isolation forest, as well as selecting features that can be done either manually or by SelectKBest() function that selects all features. The following experiment introduce the neural networks model training process, including the three essential phases. Training the data as features and labels, associating the transaction and its label in the learning phase, and finally the model is asked to make predictions about unseen data set.

Fitting the model is the first step, so, *fit()* method was called by the created model to train the data with a choice of 15 features represented by the k value. The chosen **epochs** number was 100 and the **batch_size** is set to 32 as shown in 6.14, and a sample of the results is shown in figure 6.16.

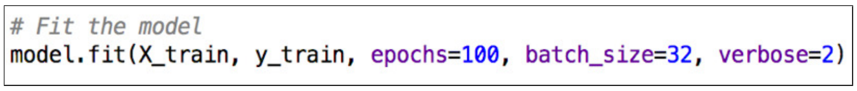
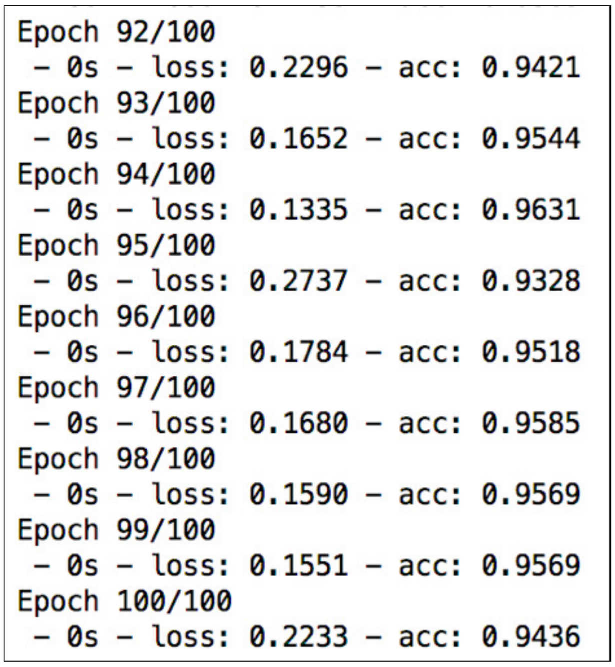

```
# Fit the model
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=2)
```

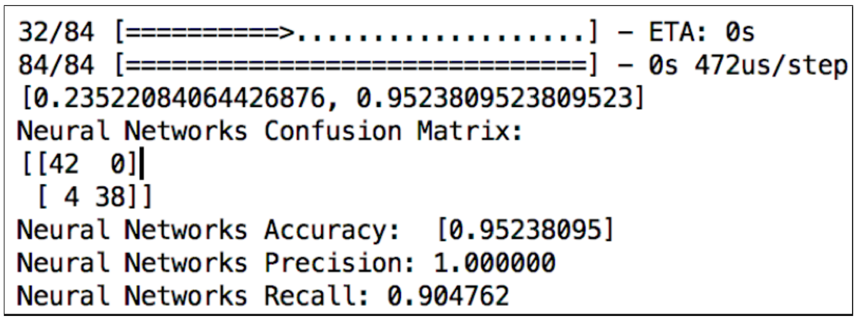
Figure 6.15: Fitting the model

The results of the loss accuracy metrics are shown as the model trained, to measure how accurate the model is during training as illustrated in figure 6.16. Also, it can be seen that the precision score achieved a perfect 1 score, and the recall scored 0.90 in the results shown in figure 6.17.



```
Epoch 92/100
 - 0s - loss: 0.2296 - acc: 0.9421
Epoch 93/100
 - 0s - loss: 0.1652 - acc: 0.9544
Epoch 94/100
 - 0s - loss: 0.1335 - acc: 0.9631
Epoch 95/100
 - 0s - loss: 0.2737 - acc: 0.9328
Epoch 96/100
 - 0s - loss: 0.1784 - acc: 0.9518
Epoch 97/100
 - 0s - loss: 0.1680 - acc: 0.9585
Epoch 98/100
 - 0s - loss: 0.1590 - acc: 0.9569
Epoch 99/100
 - 0s - loss: 0.1551 - acc: 0.9569
Epoch 100/100
 - 0s - loss: 0.2233 - acc: 0.9436
```

Figure 6.16: Model results while training



```
32/84 [==========>...................] - ETA: 0s
84/84 [==============================] - 0s 472us/step
[0.23522084064426876, 0.9523809523809523]
Neural Networks Confusion Matrix:
[[42  0]
 [ 4 38]]
Neural Networks Accuracy:  [0.95238095]
Neural Networks Precision: 1.000000
Neural Networks Recall: 0.904762
```

Figure 6.17: NN evaluation metric

However, in order to know how well a classifier performs on the data set, performance accuracy must be acquired and compared to the training performance. The performance of Neural Networks classifier in this attempt resulted in approximately 0.95 (95%) accuracy score. The training set contained 450 one instances, and 1500 zero instances, with the same distribution of the usual test set. Table 6.1 below also shows the resulted confusion matrix extracted from the results above.

CM TABLE	Predicted: 0 Transaction	Predicted: 1 Transactions
Actual: 0 Transactions	42 True Positive (TP)	0 False Negatives (FN)
Actual: 1 Transactions	4 False Positive (FP)	38 True Negatives (TN)

Table 6.1: NN Confusion Matrix Table

42 transactions were successfully predicted by the classifier as fraudulent transactions and they actually were, and there are not any false predicted fraudulent transactions. Moreover, 38 legitimate transactions were predicted as legitimate transactions correctly, and four legitimate transactions were falsely predicted as fraudulent. All of the performance evaluation techniques indicated that the obtained testing results were better than the loss accuracy resulted while training the model, and that means the classifier has learned well in the training phase and it gives better and desired outputs on the unseen test set. However, in the following sections, both load balancing and feature selection were considered to produce the evaluation metrics.

Feature Selection

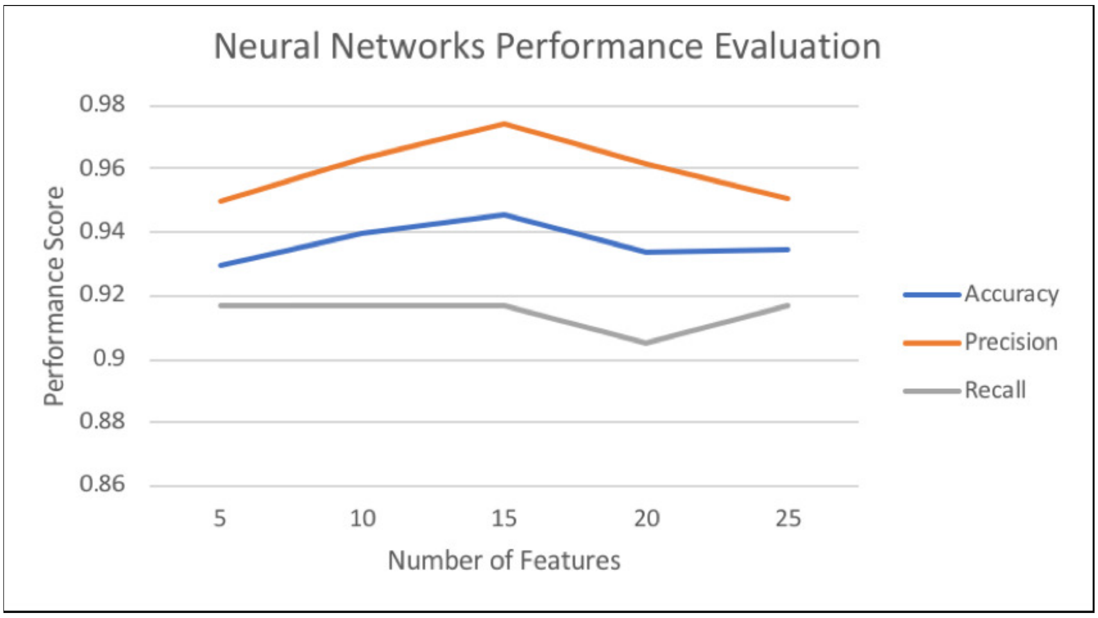


Figure 6.18: NN performance metrics evaluation

As the evaluation performance differs depending on the number of features selected using k value, the shown above figure 6.18 demonstrates an attempt to obtain the average accuracy, precision and recall in two tries on neural network models with different number features added. The accuracy score was between approximately 0.93 and 0.95. Whereas, the precision remains between 0.95 and 0.975, and recall score ranges between 0.90 and 0.91. As also indicated in figure 6.18, the highest scores were achieved using 15 features which indicates that the algorithm performs better with this selection, and the lowest accuracy achieved was using 20 features.

Load Balancing

Another experiment was taken using 15 feature selection, which is the selection used in the previous section where the highest accuracy was achieved. The experiment was done on different amount of legitimate transactions shown in figure 6.19, similarly to what was done on Random Forest and Isolation Forest. The average accuracy was obtained after 9 attempts for each different amount of instances provided, and the **batch_size** was set to 10 with 50 **epochs**.

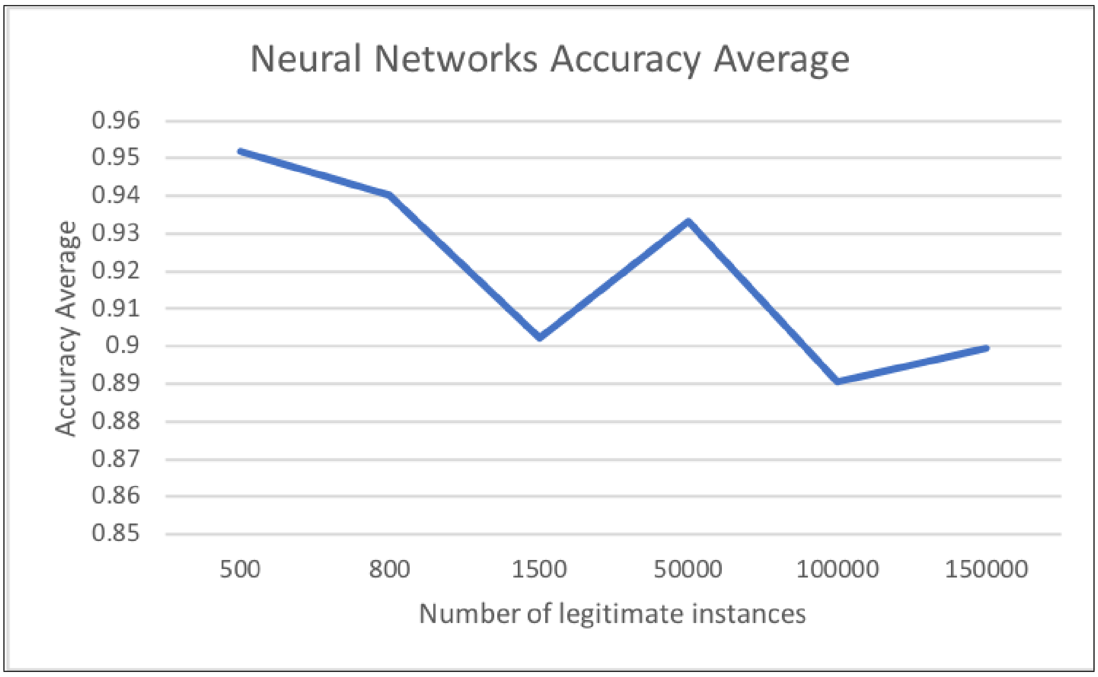


Figure 6.19: NN classifier accuracy with different amount of legitimate instances

This algorithm prefers a more balanced train set, as it is shown above, the accuracy score reached almost 0.95 with 500 instances, whereas it is dropped with choosing 800 zeros onward. However, it slightly recovers between 1500 and 50000 number of zeros instances as it achieved a 0.94 accuracy score, and then it drops again to approximately 0.89.

6.1.4 Classifiers Performance Summary

As can be seen from the above shown results and figures of all the experiment done by each classifier, some observations can be inferred,

1. The accuracy of Random forest classifier in different experiments was ranges between 0.93 and 1.00 as the highest. Whereas, the Isolation Forest classifier has reached an accuracy ranges between 0.90 and 0.94. Neural networks also achieved a high accuracy, between 0.93 and almost 0.99.

2. The graphs illustrate that whenever there is a dramatic increase in the number of data in Isolation Forest classifier, it results in a higher accuracy score ranges from 0.6 up to 0.91. Which is what was expected, due to the fact of having more data in training set results in better classifier ability to create a model for this outlier detection algorithm. In contrast, Random Forest accuracy was dropping with having a high amount of data.

3. Noticeably, the lowest accuracy achieved among all experiments and classifiers was obtained by Isolation Forest classifier, and the highest was achieved by Random Forest in the case of having a load balanced sets and all features selected.

4. The best accuracy ever reached throughout all the experiments was a perfect 1.0 by Random Forest classifier, using 800 zeros and 450 ones with all features selected.

5. Precision score remained between 0.99 and a perfect 1 in Random Forest, and between 0.89 and 0.91 in Isolation Forest, whereas in neural networks the score was between 0.95 and approximately 0.97.

6. Recall score in Random Forest remained between 0.88 and 0.90, whereas, in Isolation Forest it averages between 0.89 and 0.93 , and Neural Networks recall score was approximately between 0.90 and 0.92.

6.2 Limitations

There are many constraints in the credit card fraud detection domains that will be discussed in this section. This domain is one of the most studied domains of fraud detection and it depends on the automatic analysis of recorded transaction to detect fraudulent. One of the main challenges faced in this project was the shortage of credit card transactions data sets, and this was a wide problem associated with most of the research conducted in this particular field, as this problem was also faced by many authors, like [6]. Privacy reasons and customer sensitive details are behind the unavailability of this real data.

The given data set is has around 99% of the data set are legitimate transactions, and only around 1% fraudulent transactions. So, a main challenge of having a highly unbalanced data set was also faced. Thus, the test sets have been very small, consisting of 84 perfectly balanced samples. In order to remove bias as much as possible, the test sets were chosen randomly at each run, where the fluctuations of accuracy could be in the magnitude of 0.05, more or less. However, having a bigger data set that contains more fraudulent transactions would be better, and will guarantee more confidence in the model result obtained.

Also, fraud possibilities are increasing over time, as well as it become more sophisticated and complex due to the dynamic behaviour of fraudsters. Hence, the process become unpredictable by human experts as the behaviours and the fraud types are modified constantly along with the new detection systems created. Although these challenges are existing, many solutions of credit card fraud detection were applied as it is still an interesting and challenging field for researching.

6.3 Future Work

Due to the constraints of time and resource for an academic project, the most prioritised deliverables of this system were mainly the functional requirements, which were important to solve this problem. This system is an example of the capability of machine learning, it would need substantial development investment to become a product.

As the solution can be implemented in two different ways, either machine learning or deep learning. I decided to implement two machine learning algorithms, along with a single deep learning algorithm which is entirely based on my preference. Therefore, the first improvement could possibly be adding more classifiers such as Nearest Neighbour, Support Vector Machine and Decision Tree algorithms. This would be useful in further evaluation and testing to obtain better performance metrics in the given data set.

Furthermore, an improvement side of this software, is that it needs further optimisation techniques that could optimises automatically, such as, Grid Search which is an approach to hyper-parameters tuning, that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid. Since in the provided solution, this was done manually as much as possible. Thus, better and more accurate results can be observed using these techniques.

Additionally, the system can be improved in a way that gives the user the opportunity to change the parameters of each classifier, the k value and the number of instances used in each experiment, instead of changing them manually from the code itself and re-run the software system all over again.

A graphical user interface using different technologies like, HTML, Java Script and CSS, is a desirable improvement to enhance the user interface of the system instead of using the simple command line application. As it allows illustrating the results of testing in graphs, and allows the parameters to be shown and chosen by the user directly not from the code itself.

6.4 Conclusion

To conclude, throughout this report, multiple tests were performed on the above mentioned implemented classifiers, in order to evaluate and find the best one that can classify credit card fraudulent and legitimate transactions using machine learning and deep learning techniques. This part was solved by three different models, and evaluated using the performance metrics including accuracy, precision and recall. In terms of accuracy, and according to the results shown in the summary above, different experiments were utilised and resulted in high accuracy scores achieved by both random forest and neural networks models in different experiments. The lowest accuracy score achieved over the experiments was by Isolation Forest model. Whereas, as Random Forest achieved the highest. A simple command-line interface was also utilised to run the created engine and to evaluate each classifier independently, as the user chose. However, the accuracy of the output is linked to the quality of the training phase, which is in itself is no simple task, and yet the stated output is to a large extent inexplicable. Therefore, it cannot be decided which is most suitable classifier in both, solving the credit card fraud detection problem and predicting fraudulent transactions correctly as it depends on the data set given and the experiments applied. However, on the plus side unlocking the power of the CPU is done to continuously evaluate and re-evaluate data sets, to find things or achieve things that at this time there is no other way to do it. A very large simulated brain is used to identify patterns and then highlight them as useful results. This obviously opens up opportunities in areas such as speech recognition, big data analysis studies etc. The learning aspect here avoids the programmers of having to describe language to the computer for example, or to programmatically instruct a computer exactly what card fraud will look like based on the current knowledge.

Chapter 7

Reflection on Learning

While reflecting on the experience of writing a thesis, I came to realisation that a lot of commitment and dedication are required in order to complete the final year dissertation and implement a successful solution along with thorough report. Doing it was certainly unlike any other projects taken, thus, the learning outcomes were also different to the learning outcomes of any other project. Moreover, the main lesson taken from doing this project was about acknowledging how important is to write the report as you go along. As I implemented the software within a month and then I started to write the report, which resulted in realising while writing the report, that most of the information and details I planned to write about were no longer remembered. So, I had to go all over the things I did while implementing the software system again, and that time could have been useful in doing something useful.

Moreover, the programming language of producing the current solution was one of the hardest decisions taken. As I spent my time in implementing the software using Java object-oriented for the first two weeks, because I concerned about the minimal knowledge I had about Python, and learning more about it in such a short time. Although that I found the libraries and modules are more available in python. However, I faced a lot of difficulties using Java. So, to overcame it, I followed my supervisor's advice and start everything all over again using Python, which was more useful in machine learning approaches as it offers a huge amount of packages and libraries that are easier to learn in the timescale given.

Moreover, my minimal knowledge expanded in many aspects while doing this project. First aspect was Machine Learning and Deep Learning fields. I achieved studying and understand the algorithms used in this solution, and becoming more familiar with the libraries and classifiers implemented, constructing models, training, testing and evaluating them, finally producing a thorough report of entire product.

Another important aspect was learning LaTeX for writing academic reports. As I have been taught LaTeX previously in my first year of Computer Science degree, but I never reused it for my previous reports, as I had a little experience which I deeply regret. Since I started to write this report using Microsoft Word and experienced how much effort it requires to produce an organised output of the report, including figures, captions and mathematical equations. Therefore, LaTeX was used to produce a well organised and professional report with less efforts, and using it will be into consideration for future reports. Finally, I learnt that when implementing a software, it is always good approach to break the code into smaller parts of classes and modules, rather than coding everything in one file.

Overall, I believe that the knowledge and skills gained from this project will be beneficial in my later career which involves a department that focuses on this type of application considering the security aspect. And this project is an example of the important approaches of many security and fraud detection applications.

If you genuinely want something, dont wait for it, teach yourself to be impatient.

Gurbaksh Chahal

Bibliography

[1] Almarsoomi, H. and Kurnaz, S. 2019. Credit Card Fraud Detection Using Machine Learning Methodology. *International Journal of Computer Science and Mobile Computing,* [Online] 8(3), pp.25760. Available at: https://www.academia.edu/38608138/Credit_Card_Fraud_Detection_using_Machine_Learning_Methodology_ [Accessed 9 Apr. 2019].

[2] Asaithambi, S. 2018. *Why, How and When to apply Feature Selection* [Online]. Available at: https://towardsdatascience.com/why-how-and-when-to-apply-feature-selection-e9c69adfabf2 [Accessed 23 Apr. 2019].

[3] Breiman, L. 2001. Random Forests. *Machine Learning*, 45, 5-32 [Online]. Available at: http://dx.doi.org/10.1023/A:1010933404324 [Accessed 5 Apr. 2019].

[4] Bronlee, J. 2017. *How to Use the Keras Functional API for Deep Learning* [Online]. Available at: https://machinelearningmastery.com/keras-functional-api-deep-learning/ [Accessed: 26 Apr. 2019]

[5] Brown G. 2009. Ensemble learning. C. Sammut, G. Webb (Eds.) *Encyclopedia of Machine Learning, Springer* [Online]. Available at: http://www.springer.com/computer/artificial/book/978-0-387-30768-8. [Accessed: 7 Apr. 2019]

[6] Chen, R., et al. 2011. *Detecting Credit Card Fraud by Using Questionnaire-Responded Transaction Model Based on Support Vector Machines* [Online]. Available at: https://www.researchgate.net/publication/221252937_Detecting_

Credit_Card_Fraud_by_Using_Questionnaire-Responded_Transaction_Model_Based_on_Support_Vector_Machines [Accessed: 15 Apr. 2019].

[7] Data Robot. 2019. *What Is Unsupervised Machine Learning?* [Online]. Available at: https://www.datarobot.com/wiki/unsupervised-machine-learning/ [Accessed 3 Apr. 2019].

[8] Donges, N. 2018. *The Random Forest Algorithm* [Online]. Available at: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd [Accessed 4 Apr. 2019].

[9] Ehramikar, S. 2000. *The Enhancement of Credit Card Fraud Detection Systems using Machine Learning Methodology* [Online]. Available at: https://tspace.library.utoronto.ca/bitstream/1807/16018/1/MQ50338.pdf [Accessed 12 Apr. 2019].

[10] Ferry, S. (2017). *Delivering measurable customer benefits.* Case studies from the private and public sectors. Available at: url=https://pseconsulting.com/wp-content/uploads/2017/01/3-Pay360-Stephen-Ferry-1.pdf [Accessed 7 May 2019].

[11] Financial Fraud Action UK. 2017. *Fraud the Facts* [online] Available at: https://www.financialfraudaction.org.uk/fraudfacts17/assets/fraud_the_facts.pdf[Accessed: 1st Feb. 2019].

[12] Freund, Y. and Haussler, D. 1994. *Unsupervised learning of distributions on binary vectors using two layer networks.* Technical report, Santa Cruz, CA, USA.

[13] Goodfellow, I., et al. 2016. Deep Learning. s.l.:Massachusetts Institute of Technology.

[14] Jha, S., et al. 2012. Employing transaction aggregation strategy to detect credit card fraud. *Expert systems with applications* 39(16), pp. 1265012657.

[15] Kaggle, Machine Learning Group. 2013. *Credit Card Fraud Detection* [Online]. Available at::`https://www.kaggle.com/mlg-ulb/creditcardfraud`[Accessed 27 Jan. 2019].

[16] Keras. 2015. [Online]. Available at: `https://keras.io` [Accessed: 13 Apr. 2019].

[17] Khare, N. and Sait, S. 2018. Credit Card Fraud Detection Using Machine Learning Models and Collating Machine Learning Models. *International Journal of Pure and Applied Mathematics* 118(20), pp.825-838.

[18] Kundu, A., Panigrahi, S., Sural, S., and Majumdar, A. K. 2009. Blast-ssaha hybridiza- tion for credit card fraud detection. *IEEE Transactions on Dependable and Secure Computing*, 6(4), pp. 309-315.

[19] Lakshmi, S. and Kavila, S. 2018. Machine Learning For Credit Card Fraud Detection System. *International Journal of Applied Engineering Research* 13(24), pp. 16819-16824.

[20] Liu F.T., Ting K.M., Zhou Z.-H. 2008. Isolation forest. *Data Mining.* ICDM08. Eighth IEEE International Conference on, IEEE (2008), pp. 413-422.

[21] Maruti Techlabs. 2019. *How Machine Learning Facilitates Fraud Detection?* [Online]. Available at `https://www.marutitech.com/machine-learning-fraud-detection/` [Accessed 3 Apr. 2019].

[22] Miljkovi, D. 2010. *Review of Novelty Detection Methods* [Online]. Available at: `https://www.researchgate.net/publication/261424710_Review_of_novelty_detection_methods` [Accessed: 2nd Feb. 2019].

[23] Munoz, A. 2014. *Machine learning and optimization.* Available at: `https://www.cims.nyu.edu/~munoz/files/ml_optimization.pdf` [Accessed: 2nd Apr. 2019]

[24] Navlani, A. 2018. *Random Forests Classifiers in Python.* [Online] Data Camp Community. Available at: `https://www.datacamp.com/community/tutorials/random-forests-classifier-python`[Accessed 8 Apr. 2019].

[25] Numpy. 2005. [Online]. Available at: `https://www.numpy.org` [Accessed: 13 Apr. 2019].

[26] Pandas. 2018. [Online]. Available at: `:https://pandas.pydata.org` [Accessed: 13 Apr. 2019].

[27] Pumsirirat, A. and Yan, L. 2018. Credit Card Fraud Detection using Deep Learning based on Auto-Encoder and Restricted Boltzmann Machine. *International Journal of Advanced Computer Science and Applications* 9(1).

[28] Python. 2019. *Pickle Python object serialisation* [Online] Available at: `https://docs.python.org/2/library/pickle.html` [Accessed: 19 Apr, 2019].

[29] Saeys, Y., et al. 2008. Robust feature selection using ensemble feature selection techniques, *Proceedings of the 25th European Conference on Machine Learning and Knowledge Discovery in Databases*, Part II, Heidelberg Springer - Verlag, pp. 313-325 [Online]. Available at: `https://link.springer.com/content/pdf/10.1007%2F978-3-540-87481-2_21.pdf` [Accessed: 18 Apr, 2019].

[30] SAS. 2019. *Machine Learning: What it is and why it matters* [online] Available at: `https://www.sas.com/en_gb/insights/analytics/machine-learning.html` [Accessed 3 Apr. 2019].

[31] Scikit-learn. 2017. [Online]. Available at: `:https://scikit-learn.org/stable/` [Accessed: 13 Apr. 2019].

[32] The UK Cards Association. 2017. *UK Card Payment Summary 2017* [Online]. Available at: `http://www.theukcardsassociation.org.uk/wm_documents/UK%20Card%20Payments%202017%20-%20Summary%20FINAL.pdf` [Accessed: 2nd Feb. 2019].

[33] Tun, J. 2018. *Semi-Supervised Outlier Detection Algorithms* [Online]. Available at: `https://cloudfront.escholarship.org/dist/prd/content/qt1f03f6hb/qt1f03f6hb.pdf?t=p621x4`[Accessed 6 Apr. 2019].

[34] Winham, S. J., et al. 2013. A weighted random forests approach to improve predictive performance. *Statistical Analysis and Data Mining: The ASA Data Science Journal,* 6(6), pp. 496-505.

[35] Witten, I. et al. 2016. *Data Mining: Practical Machine Learning Tools and Techniques.* Elsevier.