The University Of Sheffield.

Aerospace Engineering

Final Project Report Cover Sheet

Surname: Ogbonnaya Forenames: Samuel Iheanyichukwu

Registration number: 130163198

Module code: COM 488

Module name: Aerospace Individual Investigative Project

Assignment title: Application of Machine Learning Classifiers for Predicting the Outcome of Bank Telemarketing

Supervisor: Professor Eleni Vasilaki

Date submitted: 08/05/2018

Student's signature: SAM.O

Date: 08/05/2018

The University Of Sheffield.

Aerospace Engineering.

Aerospace Engineering
Final Year Project

Application of Machine Learning Classifiers for Predicting the Outcome of Bank Telemarketing

Samuel Ogbonnaya

May 2018

Professor Eleni Vasilaki and Professor Roger Moore

Dissertation submitted to the University of Sheffield in partial fulfilment of the requirements for the degree of
Master of Engineering

Abstract

Telemarketing campaigns are often employed by banks and other financial institutions for the purpose of increasing the sales of their products to prospective clients. On some occasions, this technique may not be received well by clients due to privacy intrusion. Thus, the institution is required to re-direct its efforts to clients which are more likely to be receptive to the marketing campaign. Machine learning techniques can be used to achieve this objective by predicting the likelihood of a prospective client buying a product.

In line with the aforementioned, this study investigated the performance of several parametric and non-parametric classifiers namely, Gaussian Naïve Bayes, Logistic Regression, Decision Trees, Linear SVC and Multilayer Perceptron Artificial neural network on classifying prospective clients using only priori data from the bank marketing dataset. Voting and Adaptive Boosting ensemble methods were also evaluated on improving the performance of the individual classifiers listed above.

The results of this study show that for this particular problem, the parametric classifiers perform slightly better than the non-parametric methods due to less overfitting. The ensemble methods do indeed improve upon the performance of the individual classifiers with the Adaptive Boosting classifier achieving an AUC of 0.7476 with only priori information. With posterior information included for comparison purposes, the AdaBoost achieved an AUC score of 0.8825, ranking 3rd place (5% from first place) amongst the surveyed literature which evaluated against the AUC metric and used posterior information.

Table of Contents

List of Figures

List of Tables

Acknowledgement

First and foremost, I would like to thank God for granting me this opportunity and for his guidance throughout this project. I would also like to thank my Supervisor, Professor Eleni Vasilaki for her steadfast guidance and support throughout this project. My Parents; Mr Samuel and Mrs Ngozi Ogbonnaya for all their love and support. My sisters, Chinenye and Ezinne Ogbonnaya for their prayers and unrelenting motivation. Finally, my friends, namely Megan Picton and Femi for their constant support throughout my project.

1 INTRODUCTION

In financial institutions, future insight on major business activities such as sales, marketing, logistics etc. is highly beneficial to the C-level management and their subordinates. This future insight enables the management to make more effective strategic business decisions, thereby achieving targets such as increased profitability, customer loyalty and cost reduction.

Indeed, financial institutions place a significant emphasis on marketing in order to positively impact earnings and increase customer base. According to the research conducted in [2], effective promotion of superior products can lead to successful launches of new products and services, and by building good customer relationships, a banking organisation may increase its financial earnings [3].

Direct marketing is a technique employed by financial institutions to promote their product and services and also develop and maintain good customer relationships [4]. When managed properly, direct marketing is effective and can build brand loyalty [5]. However, a major issue with direct marketing is a perceived threat to customer privacy and image impact on institutions [5], this results in ineffective and costly marketing campaigns. In order to mitigate this concern, a countermeasure can be employed to create a targeted campaign for a select group of customers who are most likely to positively respond to the campaign [5], thereby reducing the probability of privacy intrusion and also reducing marketing budget waste.

A mechanism to determine this target group of customers (i.e. customers who will respond positively to a promoted financial product) is therefore beneficial for financial institutions. Machine learning models can be used to make predictions on how a customer with certain features will respond to a marketing campaign – either 'positive' or 'negative'. With the results derived from these predictions, marketing leaders will gain better insights as to which type of customers will most likely respond positively and can therefore target the right customers to increase their earnings, decrease marketing cost and avoid privacy intrusion.

In this work, the performance of parametric and non-parametric classification algorithms will be evaluated to deduce the best methods. Feature selection and ensemble methods will also be evaluated for the purpose of improving the performance of the classifiers. The bank marketing dataset obtained from the UCL Machine Learning Repository [1] is utilised for this work, the financial product marketed was a bank term deposit in which the clients either 'Subscribed' or did 'Not subscribe'.

It must be noted that the '***duration***' feature in the bank marketing dataset was **not** utilised in this work. This is because this attribute highly affects the output target and in addition, prior to contacting a client, the call 'duration' is not known, therefore the inclusion of this attribute leads to unrealistic models [1], [11] and [45]. The exclusion of this attribute brings some novelty to this work as other research have included this attribute.

1.1. Aims & Objectives

The key aims of this research project are to:

1. Investigate the performance of diverse machine learning classifiers to realistically and effectively predict if a bank client will subscribe to a new bank product without the use of information regarding the contact execution i.e. **'duration'**. Therefore, the objectives will involve:
    a) Identifying optimal feature sets for representing bank marketing data.
    b) Comparing the performance of parametric & non-parametric algorithms using an appropriate evaluation metric.

2. Consider the effect of ensemble methods on improving algorithm performance due to the scarcity of research that utilise ensemble methods for the bank marketing data set. Therefore, the objectives will involve:
    a) Combining the classifiers evaluated in the first part to create a new classifier with superior performance using Voting and Adaptive Boosting techniques.

In order to achieve the aims of the project highlighted above, the following tasks will need be conducted:

- Division of the Bank Marketing Dataset into a training set for training the algorithms, a validation set for regularization and a test set to measure final performance of the algorithms.
- Pre-processing of the dataset subset using appropriate techniques through feature engineering etc.

- Selection of a set of algorithms and training parameters to evaluate, making an informed decision based on the following:
    - The linear separabilty of the data
    - The problem type – classification.
    - Methods implemented in the surrounding bank marketing literature
- Evaluating ensemble methods for increasing performance.

1.2 Overview of Chapters

The remainder of this paper continues with an overview of various key concepts and a review of the literature surrounding the bank marketing classification problem in Chapter 2, this chapter is aimed to develop the understanding of the reader and will also be referenced throughout the remaining sections. Chapter 3 presents the design methodology along with how it was implemented to achieve the aims and objectives stipulated in section 1.1. The results of the experiments are presented and discussed in Chapter 4, followed by concluding remarks and additional work in Chapter 5.

2 BACKGROUND AND LITERATURE REVIEW

In this chapter, we shall explore the fundamental principles which govern the implementation of machine learning algorithms on real-world data. The background and concepts covered in this section are structured in line with the workflow required to execute a typical machine learning project.

2.1 Machine Learning

The aim of machine learning is to program a computer to optimize a performance criterion with respect to a particular problem by using training data or past experience [6]. Since its inception, machine learning has been used to develop solutions to a number of problems in industries such as manufacturing, finance, retail, telecommunication and medicine etc. For example, in railway operation management, research in [7] shows the use of a large amount of historical data to develop a model for predicting conditions which lead to failure in railway systems, thereby helping prevent service interruptions. In financial applications, [8] shows the effectiveness of applying machine learning to determine the probability of default of credit card clients.

The range of problems to which machine learning is applied to can be broadly categorized into the archetypes defined below. The categorization of these archetypes is largely dependent on the nature of the dataset used in a task.

2.1.1 Supervised Learning

In this problem type, the training dataset utilized is 'labelled', that is, for each instance of input variables there is a known output variable. If we take an input, X and an output, Y, the goal of the machine learning algorithm is to utilise training data to derive a function, t which maps the output, Y from input X as shown below:

$$t: X \rightarrow Y \tag{2.1}$$

The parameters of this function will be such that, the approximation error between X and Y will be minimized in order to obtain predictions which are close as possible to the correct values from the training data.

2.1.2 Unsupervised Learning

Here, the training dataset is ‘unlabelled’. Algorithms of this type are employed to infer a function to elucidate unseen clusters or groupings of data instances in the training dataset which occur more regularly than others, this is known as density estimation [9]. Typical unsupervised learning problems are clustering, dimensionality reduction etc.

2.1.3 Semi-supervised Learning

As may be inferred from the name, Semi-supervised learning algorithms are based on training datasets which contain both ‘labelled’ and a large amount of ‘unlabelled’ data. This situation occurs due to labelled instances being difficult and expensive to acquire in comparison to ‘unlabelled’ data [10]. This type of learning is made possible by formulating a model assumption that the clusters of ‘unlabelled’ data should have similar labels to nearby ‘labelled’ data [10].

2.2 The Bank Marketing Classification Problem

Between May 2008 and November 2010, a Portuguese retail bank conducted telemarketing campaigns to elicit bank clients to subscribe to long term deposits. The outcome for each call to a client was collected and consolidated into a large database which is available from the UCI Machine Learning Repository [1].

The collected data specifies client personal information, social and economic information and the campaign information with approximately 41188 instances. It is multivariate in nature and includes 20 input variables such as customer age, profession, education, mortgages, credit cards etc. with a numeric or categorical format. The output variable, desired target (‘subscribe’ or ‘not subscribed’) is of a binary format.

As stated previously in section 2.1, machine learning is a mechanism which can be used to conduct predictive data analysis. In the business context set above, the primary objective of applying machine learning will be to:

- Increase the efficiency of the telemarketing campaign by decreasing the number of clients contacted.

- Increase the number of long-term deposits sold by targeting clients who are most likely to subscribe to the long-term deposit.

These two objectives can be achieved by having highly accurate knowledge of bank clients who will either ‘subscribe’ or ‘not subscribe’ to the long-term deposit.

Based on the learning types discussed previously and the context set above for this telemarketing campaign in which labelled data is provided. The bank marketing problem is therefore characterized as a classification problem and will require the use of ***supervised learning algorithms.*** These algorithms will be discussed in section 2.4 and the processing techniques for the data prior to using the algorithms will be discussed in section 2.3.

2.2.1 Feature Requirements

A key attribute in this bank marketing dataset is the **‘duration’** attribute. According to [1], [11], and as shown in [45]; this attribute is of great importance in the classification of the bank marketing data set. It ranks first in the Information Gain and Chi-square feature importance tables in [45], thereby concluding its significance in classifying the output target. However, ‘duration’ is a posterior attribute i.e. its value is not known prior to making a call, thus, the inclusion of this attribute for developing predictive models is unrealistic.

To conduct a novel and realistic approach to the classification problem, this attribute should not be utilised in the implementation of the predictive models. However, it must be noted that the exclusion of the attribute will lead to a more difficult classification problem [11, 45] this is intuitive as a highly important classification feature is removed.

2.3 Data Pre-processing

Unprocessed real world data generally contains unreliable and noisy information which can make classification strenuous and adversely impact the classification performance of an algorithm. To obtain optimal performance from a machine learning algorithm on a given task, input data should be initially processed before it is fed into a model. In this sub-chapter, we shall review common data pre-processing and visualisation techniques which are applied in machine learning.

2.3.1 Dimensionality Reduction

For most machine learning algorithms, the complexity of a classifier is proportional to the number of inputs dimensions and the size of the utilized training dataset [9]. Datasets with high dimensionality characteristics use up more memory and significantly increase computational time for training algorithms. In addition, real-world data is expensive and difficult to extract, it is therefore of interest in a machine learning project to reduce the dimensionality of a problem, this is known as Dimensionality Reduction [9].

Dimensionality reduction is defined in [29] as the procedure of finding intrinsic low-dimensional structures hidden in the high-dimensional observations. Reducing dimensionality leads to simpler models with less variance to dataset characteristics such as outliers or noise. These simpler models thereby reduce the time required for computation. However, it must be noted that dimensionality reduction may lead to loss of relevant information as it involves the removal of data. A systematic approach should be used when this process is carried out to mitigate information loss and ensure only redundant data is removed. Two main methods which are common for tackling the dimensionality reduction problem are feature selection and feature extraction [9]. A high-level overview of these methods are described below.

Feature Selection

Feature selection as the name implies involves removing a subset of the available features in a dataset for the purpose of reducing computational time and increasing performance. Common approaches for implementing feature selection to a problem are the filter approach and wrapper approach.

- ***Filter Approach***

The filter approach utilises the properties of the dataset for selecting relevant features. A scoring mechanism is usually applied, such that a relevance score is calculated for each feature, and the highest scoring features will be selected [31]. The technique is advantageous for problems with high dimensional datasets and when multiple algorithms need to be evaluated because it is independent of the classification algorithms and therefore only needs to be applied once. However, it may not be as effective for datasets which contain features with strong correlations. This method treats each feature independently and is therefore unable to account for feature dependencies [31]. Applications of this method in literature include, [17] and [18], where a decision tree algorithm was applied to the training dataset to select relevant features.

- ***Chi Square Test***

This method can be categorized under the Filter approach for feature selection. Chi-square test measures dependence between stochastic variables. Utilising the chi-square test for feature selection removes features which are most likely to be independent of target class and therefore irrelevant for classification. The chi-square formula adopted from [72] is shown in equation 2.2 below:

$$x_c^2 = \sum_{i=0}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{2.2}$$

With $(r-1) * (c-1)$ degrees of freedom.

Where O_i is the observed count, r is number of rows, c is number of columns, and E_i is the expected counts.

- ***Wrapper Approach***

As opposed to the filter techniques described above, the wrapper approach does not treat feature independently, but incorporates the feature selection with the model selection, essentially ‘wrapping’ the search algorithm around the classification model. A search across the entire feature selection space is carried out and various feature subsets are created and evaluated. This search method may be randomized or deterministic. The deterministic wrapper approach recursively adds or removes features until a termination criterion is met [35]. When features are added to an empty set, this is known as a forward selection and when they are removed from a full set, this is known as backward selection [35]. A backward selection strategy was applied in [11] to aid with increasing the model performance, whilst in [12], Moro et al applied a two-step feature selection approach which utilized the business domain knowledge and an adapted forward selection to reduce the number of features. Disadvantages of the wrapper approach include susceptibility to overfitting and increased computational intensity [35]. In addition, obtaining an optimal feature subset will require an exhaustive search across the feature space, and this is very expensive to conduct [36].

Feature Extraction

Feature extraction can also be utilised to tackle the high-dimensionality problem. It involves finding new subsets of the original dataset and combining these to create new feature set. Feature extraction techniques could be supervised or unsupervised depending on if they utilise the target information [9].

Across machine learning literature, commonly known linear methods for feature extraction are Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) and non-linear methods include Locally Linear Embedding (LLE). In depth details of these algorithms can be found in [9].

2.3.2 Normalization

Normalization is an important pre-processing technique under feature scaling which transforms input variables into a certain range, e.g. 0-1 [39]. It is implemented using the following equation:

$$X_2 = \frac{X1 - Mean\,Value}{Max\,Value - Min\,Value} \tag{2.3}$$

In equations 2.3, X_1 signifies the old feature whilst X_2 is the new transformed feature. Algorithms such as Neural networks, Support Vector Machine, Logistic Regression etc. often utilise numerical and scaled inputs e.g. range [-1, -1] and will therefore utilise normalization for feature scaling.

2.3.3 Linear Separability

Let X_0 and X_1 be two sets of points in an *n*-dimensional Euclidean space. Then X_0 and X_1 are linearly separable if there exists n + 1 real numbers $w1, w2, \dots\dots, wn, k,$ such that every point x ϵ X_0 satisfies:

$$\sum_{i=1}^{n} w_i x_i > k \tag{2.4}$$

and every point x ϵ X_1 satisfies:

$$\sum_{i=1}^{n} w_i x_i < k \tag{2.5}$$

where x_i is the $i - th$ component of x.

It is beneficial to determine the Linear separabilty of the training dataset as this will influence the type of algorithm one chooses to implement for a classification problem.

Different methods may be used to test the linear separabilty of a training dataset. [20] presents five methods based on linear programming, computational geometry, neural networks, quadratic programming, and the Fisher linear discriminant method for testing linear separabilty between two sets of points.

Aspects such as the complexity level, the difficulty of the classification problem, the ease of implementation should be considered when choosing a method for testing linear separability for a given problem [20]. The research concluded that the simplex method, perceptron model, SVM with kernels, the Fisher linear discriminants and the convex hull algorithm (datasets with $\leq$ 3 dimensions) are efficient algorithms with lower computational complexities in contrast with the other methods i.e. Fourier-Khun elimination method, Class of linear separability method.

2.4 Classification Algorithms

As stated in Section 2.1, supervised learning algorithms learn from training data and derive an inferred function (hypothesis) which maps an input, X to output, Y. The set of assumptions which are made to enable a supervised learning algorithm develop a unique solution is known as inductive bias [9]. For a given problem, the right inductive bias should be selected such that the model trained on the training data set will predict the correct result for new unseen data instances, i.e. generalizes well.

The functions of most learning algorithms take the form of a conditional probability model, P (x|y) or a joint probability model, P (x, y). These functions are generally chosen by either using empirical risk minimization or structural risk minimization [21].

Empirical risk minimization principle states that the supervised learning algorithm should choose a hypothesis which minimizes the empirical risk, $Remp$. Formal definitions of empirical risk are covered in [21].

Structural risk minimization seeks a function J(g) which minimizes a consolidation of the empirical risk, $Remp$ and a regularization penalty. i.e.

$$J\ (g)\ =\ Remp\ +\ \alpha\ *\ model\ complexity\ [9]. \tag{2.6}$$

Where $\alpha,$ is the regularization penalty weight.

If the regularization penalty weight, α is minimized, the model will be more complex with decreased bias (high variance). However, when it is maximized, bias is increased (variance decreased) and only simpler models are allowed. High bias (low variance) constitutes to a large error between our approximated function and the true function, where our approximated function erroneously maps the input to the output, this phenomenon is known as ***under fitting*** [9]. When the approximated function also learns random noise in the training data, this is known as ***overfitting*** and is thus the model has a high variance (low bias) [9].

An optimal model will be one with both low variance (good generalization to unseen data) and low bias (capture regularities in the data). In most cases, optimizing the bias of a model adversely affects the variance and vice versa. This dilemma is known as the bias-variance trade-off [22] and is a key aspect of supervised learning algorithms. Figure 2.1 adopted from [38] shows a visualisation of this problem and the characteristic of an ideal model.

There are a variety of algorithms which may be implemented for classification. Some factors which influence the selection of algorithms include:

- The nature of the input data i.e. if it contains features with varying data formats such as numeric, categorical, discrete, continuous etc. Some algorithms are more congruent with numeric features.
- The linear separabilty of the data. If the training dataset is not linearly separable i.e. algorithms such as Decision trees and neural networks are best employ as they are designed for non-linear problems.

Different classification algorithms have different approaches for estimating classes for a given dataset. The underlying approaches of many algorithms can be characterized as either

Parametric and Non-parametric methods [9]. A brief overview is presented below which describes these methods and highlights some of the algorithms with utilize each approach. Furthermore, classification algorithms may be combined using different methods, the group of algorithms which have been combined is termed an ensemble. Ensemble methods are also presented.

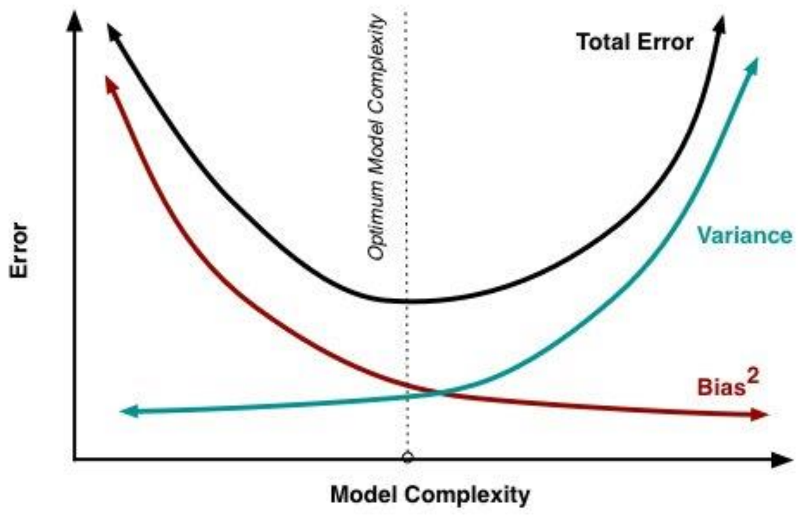

Figure 2.1: Depiction of the Bias-Variance Dilemma [38]

2.4.1 Parametric Classifiers

Parametric classifiers make decisions using information provided from the training sample. These classifiers assume that the sample fits a probability distribution which obeys a known model that can be parametrized by a finite number of parameters.

By estimating the parameters of this distribution using the training sample, and integrating them into the model, the model can be used to make a classification decision. Maximum Likelihood estimation and Bayesian estimation are common methods used to estimate distribution parameters. These methods are detailed in [9].

The review of various literature for machine learning applied to the bank telemarketing dataset show that ***Logistic regression*** and ***Naïve Bayes*** are common parametric classifiers which are employed for classification. A high level overview of these two algorithms is provided below.

Logistic Regression

The logistic regression model operates a smooth nonlinear logistic transformation over a multiple regression model [12]. The central mathematical concept that underlies logistic regression is the logit i.e. the natural logarithm of an odds ratio [26]. The model predicts the logit transformation of the output variable, Y by generating parameters of the distribution function using the training examples, X. The logit transformation is the natural log of the ratio of the output occurring to the ratio of the output not occurring. It has the form:

$$logit(Y) = ln(odds) = ln\left(\frac{\pi}{1-\pi}\right) \tag{2.7}$$

Where π, is P(Y|X), this is the Logistic Regression model.

Looking at equation 2.7, as the probability goes down to zero the odds approach zero and the logit approaches $-\infty$. At the other extreme, as the probability approaches one the odds approach $+\infty$ and so does the logit. Thus, logits map probabilities from the range (0, 1) to the entire real line and is equivalent to a sigmoid function.

Curtiz Lentz in [13] and Keles et al. [23] employed a Cross-industry Standard process for Data mining methodology for predicting the success of bank telemarketing. The research in [13] concentrated on using four different Logistic Regression models with variations of PCA components and misclassification costs to classify clients. The best model obtained a prediction accuracy of 90.09%. In another paper [23], which attempted classification on the bank marketing dataset, an accuracy of 81.2% was obtained for the logistic regression model. It made no mention of any pre-processing techniques employed for the data. Moro et el also considered a Logistic regression model in [12] for classifying clients. The results obtained show an AUC of 0.715. [24] also utilizing PCA obtained results of 0.788. Comparing both papers, [24] utilised PCA as a pre-processing step and also backward feature selection whilst only feature selection was employed in [12]. As there are many factors and variables which go into tackling machine learning problems, the difference in the results for the same model could be due to a number of reasons, one such reason could be the lack of/poor implementation of data –pre-processing.

Naïve Bayes

Naïve Bayes classifier is a probabilistic classifier based on the application of Bayes' theorem with a conditional independence assumption that all attributes are independent given the value of the class variable [25].

Zhang et al. in [25] formally defined the Naïve Bayes classifier as:

$$Y_{nb}(E) = arg\max_{y}(p(y)\prod_{i=1}^{n} p(x_i|y) \tag{2.8}$$

where, equation 2.8 assumes that $X_1, X_2, \ldots, X_n$ are n attributes of a training sample. An example E is represented by a vector $(x1, x2, \ldots\ldots, xn)$, where x_i is the value of X_i. Y represents the class variable, which takes a positive or negative value and y represents the value that Y takes.

Zhang et al [25] stipulates that Naive Bayes is easy to construct because $p(x_i|y)$ can easily be derived from the training data and has been shown to have good classification performance. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality [53].

Keles et al. [23] also compares a Naïve Bayes model amongst others for implementing a decision support system. The research stipulates a prediction accuracy of 69.14%. This may be once again due to a limited pre-processing stage. Moro et al. [11] which also utilized a Cross-industry Standard process for Data mining methodology, obtained an average accuracy of 87% after several runs, a better result in comparison.

Overall, parametric classifiers tend to produce good classification results as seen from the results above. Due to their use of a finite number of parameters, they tend to be simple, highly interpretable and fast. However, they are constrained to a specified form and therefore offer less flexibility if a training sample has a complex underlying function. If the assumption of a parametric model does not fit, the model will be very likely to generate erroneous classifications decisions.

2.4.2 Non-Parametric Classifiers

In comparison to parametric classifiers, non-parametric classifiers make no assumptions on the form of the training sample. The classifier utilizes the training data to derive a model which can be parametrized by an infinite number of parameters. They operate by determining similar past instances in the training data and interpolate between certain distance measures to 'emit' a classification decision.

Decision Trees, ***Support Vector Machines*** and ***Neural Networks*** are common examples of non-parametric algorithms which have been utilised in previous research and shown to have good performance. Further information on these algorithms are detailed in [27], [28], [29], [30], [31] & [32]. A high-level overview of these algorithms are presented below.

Artificial Neural Network

This a parallel processing structure which consists of mathematical processing elements (neurons) which are interconnected with unidirectional signal channels (weights). The artificial neuron has one or more binary inputs (on/off) and one binary output, and simply activates its output when one of its inputs are active.

- ***Perceptron***

This is one of the simplest Artificial Neural Networks architectures based on a Linear Threshold Unit (LTU). The LTU (Figure 2.2) computes a weighted sum of its inputs ($z = w1x1 + w2x2 + \cdots + wnxn = \boldsymbol{wT} * \boldsymbol{x}$), then applies a step function to that sum and outputs the result shown in equation 2.9.

$$hw(\boldsymbol{x}) = step(z) = step(\boldsymbol{wT} * \boldsymbol{x}) \text{ [39].} \tag{2.9}$$

The step function utilised in the perceptron can take the following form [39]:

$$sgn(z) = \begin{cases} -1 \; if \; z < 0 \\ \;\; 0 \; if \; z = 0 \\ +1 \; if \; z > 0 \end{cases} \tag{2.10}$$

For binary classification, a single LTU can be utilised, such that it computes a linear combination of the inputs and if the result exceeds a threshold, a positive class is returned, otherwise, a negative class is returned [6], [39]. The decision boundary (equation 2.10) for each output neuron is linear, therefore the perceptron is unable to learn complex underlying patterns. However, if the dataset being utilised is linearly separable, the perceptron will be able to converge to a solution [6].

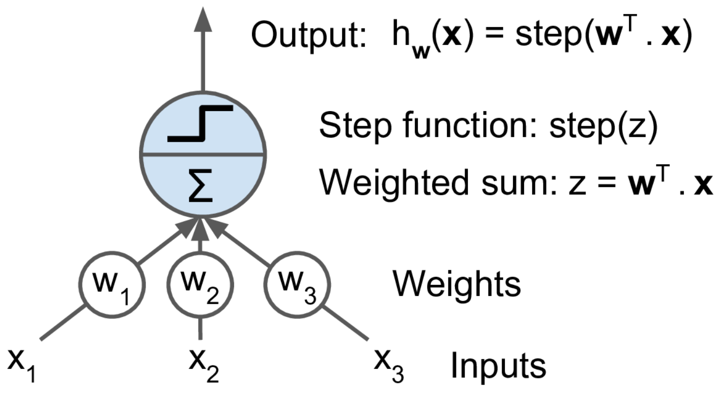
Output: $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(\mathbf{w}^T . \mathbf{x})$
Step function: step(z)
Weighted sum: $z = \mathbf{w}^T . \mathbf{x}$
Σ
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

Figure 2.2: Linear Threshold Unit. Adopted from [39]

The training of a perceptron is carried out using the ***perceptron training rule*** (weight update) [6] which revises the weight w_i associated with input x_i according to the rule shown in equation 2.11 [6]. When the perceptron is fed an instance at a time, it makes a prediction for each instance. For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction.

$$w_i \leftarrow w_i + \eta(y - \hat{y})x_i \tag{2.11}$$

Where, $\hat{y}$ is the output generated by the perceptron for the current training instance, y is the target output neuron for the current training instance and, ŋ is the learning rate.

The objective of the *perceptron training rule* is to select a weight vector **w** which will minimize the perceptron cost function over the whole training set. The cost function adopted from [6] is

$$J(\boldsymbol{w}) \equiv \frac{1}{2}\sum_{x \in X}(y_x - \hat{y}_x)^2 \tag{2.12}$$

Where X is the set of training examples, y_x is the target output of the training example and $\hat{y}_x$ is the output of the linear unit of training example x.

Gradient Descent

The selection of the weight vectors for minimizing the cost function (equation 2.12) can be optimized by using an appropriate optimization algorithm. [6] suggests the utilization of *gradient descent* search, which starts with an arbitrary initial weight vector, then repeatedly modifies it in small steps (learning rate). At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface. This process continues until the global minimum error is reached. The direction of the steepest gradient is calculated by computing the derivative of J with respect to each component vector **w**, this derivative is known as the *gradient of J,* shown in equation 2.13.

$$\frac{\partial J}{\partial w_i} \equiv \frac{1}{2}\sum_{x \in X}(y_x - \hat{y}_x)(\,-x_{ix}) \tag{2.13}$$

When this gradient, $\frac{\partial J}{\partial w_i}$ is combined with the perceptron training rule, this yields the *weight update rule for gradient descent*

$$\Delta w_i = \eta \sum_{x \in X}(y_x - \hat{y}_x)x_{ix} \tag{2.14}$$

The full derivations for equations 2.13 and 2.14 along with the procedure for the gradient descent algorithm are outlined in Section 4.3.3.2 of [6].

Although the gradient descent algorithm is a good strategy for optimizing cost functions, it suffers from a few difficulties [6]:

- it can take a long time to converge to a local minimum and.
- there is no guarantee of converging to the global minimum.

Stochastic Gradient Descent

The *stochastic gradient descent* provides an alternative optimization of the weight vector search to avoid the gradient descent issues specified above. As opposed to computing the weight updates after summing over the whole training examples in X, stochastic gradient descent approximates the gradient descent search by incrementally updating the weights after calculating the cost for *each* individual example, x_i. The *stochastic gradient descent weight update rule* is thus given by

$$\Delta w_i = \eta(y - \hat{y})x_i \tag{2.15}$$

With the cost function defined as follows:

$$J(\boldsymbol{w}) \equiv \frac{1}{2}(y_x - \hat{y}_x)^2 \tag{2.16}$$

See section 4.3.3.3 of [6] for more details.

Adaptive Moment Estimation (Adam)

Adam is another alternative for stochastic optimization of the perceptron cost function. It only requires first-order gradients with little memory requirement and computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients [60]. Some of Adam's advantages are that:

- the magnitudes of parameter updates are invariant to rescaling of the gradient
- its step sizes (learning rate) are approximately bounded by the step size (learning rate) hyper parameter,
- it works with sparse gradients, and
- it naturally performs a form of step size annealing.

Algorithm 1 gives the pseudo-code for the Adam optimization algorithm. A detailed explanation of the algorithm can be found in section 2 of [60].

- ***Multilayer Perceptron***

The lack of robustness of the perceptron to non-linear decision surfaces as highlighted above can be overcome by stacking multiple perceptron's together. The result of this combination is known as a Multilayer Perceptron. The architecture of a multilayer perceptron is in the form of layers, and typically consists of at least three layers. Input layer for input variable(s), hidden LTU layers (consisting of at least one layer) for processing and an output layer to output of the classification decision [27], [28]. A generic architecture of a Multilayer Perceptron with 3 inputs, 1 hidden layer and 2 outputs is illustrated in figure 2.3 below.

Neural networks often achieve very good classification performance. However due to the complex nature of its architecture, it is usually difficult to interpret how it makes predictions.

ALGORITHM 1: Adam Optimizer

```
1. **Require:** Stepsize
2. **Require:** Exponential decay rates for the moment estimates
3. **Require:** Stochastic objective function with parameters $\theta$
4. **Require:** Initial parameter vector
5. Initialize 1st moment vector
6. Initialize 2nd moment vector
7. $t \leftarrow 0$ (Initialize time step)
8. **while** $\theta_t$ not converged **do**
      a. $t \leftarrow t + 1$
      b. Get gradient with respect to stochastic objective at time step
      c. Update biased first moment estimate
      d. Update biased second raw moment estimate
      e. Compute bias-corrected first moment estimate
      f. Compute bias-corrected second raw moment estimate
      g. Update parameters
9. end while
10. **return** Resulting parameters
```

Models of this nature are known as black-box models. This issue can be overcome by utilising countermeasures such as sensitivity analysis and rule extraction [12].

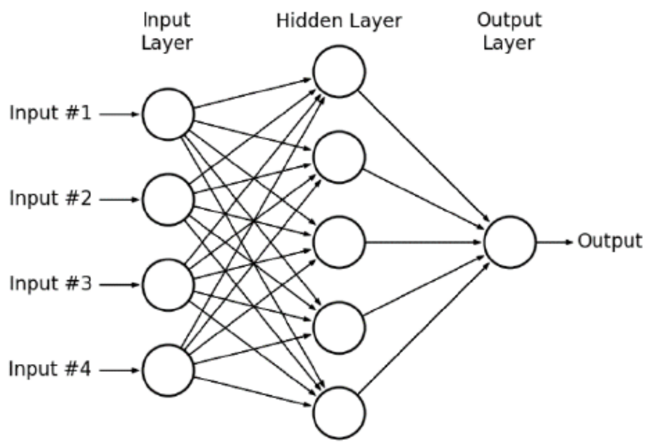

Figure 2.3: Generic Architecture for a Multilayer Perceptron. Adopted from [55]

Support Vector Machines

The Support Vector Machine is a binary linear classifier that optimizes a classification hyperplane between the surface data points of two clusters in the data space [41]. For a training data set which contains input vectors, $x_i$which are paired with the target class y_i, where $y_i \in \{1, -1\}$, Support Vector Machine seeks to find a hyperplane which maximises the marginal distance between the target class variables, i.e. where one side of this hyperplane contains data points of class 1 [o] and the other class -1 [x]. A simple visualization of the SVM classification is shown in figure 2.4.

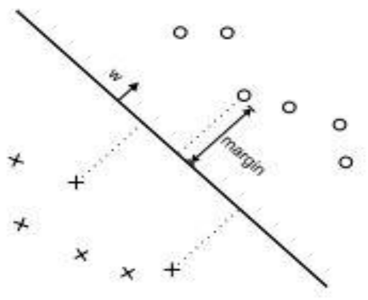

Figure 2.4: Maximum Margin separating Hyperplane for SVM adopted from [41]

The closest data points on both sides have most influence on the position of this separating hyperplane and are therefore called *support vectors* [40]. The separating hyperplane is given as:

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0 \tag{2.17}$$

where · denotes the scalar product, **b** is the bias or offset of the hyperplane from the origin in input space, **x** are points located within the hyperplane and the normal to the hyperplane, the weights **w**, determine its orientation [40].

The SVM classifies inputs by predicting their category based on which side of the separating hyperplane they fall into. SVM requires full labelling of data, therefore SVM may not be applicable to real world data sets which often have missing features. This can be countered by utilising a pre-processing technique to derive these feature. This is known as imputation [9].

Decision Trees

Decision trees are defined as a "classification procedure that recursively partitions a data set into smaller subdivisions on the basis of a set of tests defined at each branch (or node) in the tree." [32]. Figure 2.5 below shows a Generic Decision Tree.

Decision trees classify input instances by assigning them to classes based on the leaf nodes to which the instances align to. Mathematically, decision trees are rooted binary trees which begin from an initial node, a root node and then recursively splits into two branches until a stopping criteria is used [42]. The core of a decision tree algorithm lies within how a node is split, a measure known as an impurity measure is utilised for this process, where the ultimate goal is to find a split which maximises the decrease in impurity.

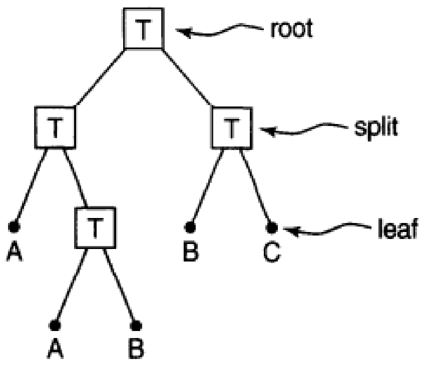

Figure 2.5: Generic Decision Tree [32]

A common impurity measure used in decision tree algorithms is the *Gini Impurity:*

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2 \tag{2.18}$$

Where $p_{i,k}$ is the ration of class k instances among the training instances in the ith node.

Decision trees are impervious to datasets with missing values in contrast to SVM. Numeric and categorical inputs can be fed into DT models without need for data transformation. This is also in contrast to model such as neural networks or SVM which require data transformation such as normalization. Decision trees is not a black box model, as its structure is unambiguous and is therefore easily interpretable.

Comparison of Non-Parametric Methods

Moro et al in [12] were one of the first cohorts to utilize the bank marketing dataset for research. They considered an Artificial Neural Network (ANN) method for classifying clients. Feature selection was applied and the number of hidden node was selected using a heuristic approach were the value was chosen as half of the number of inputs. In this model comparison research, the ANN achieved the best performance with an AUC of 0.794 when compared with Logistic trees (0.715), Decision Trees (0.757) and Support vector machine (0.767).

Moro et al as conducted research in [11], in this case, better performance was attained by the SVM with an AUC of 0.938, with DT achieving an AUC 0.868. However, the research utilised the duration attribute and concluded that the optimal model provides an explanation for marketing campaigns which have already occurred in the past.

Kim et al in [14] utilised a Deep Convolutional Neural Network for classification and obtained a classification accuracy of 79.7%. In [15], Salim Lahmiri implemented a two-step system which included several neural networks, primarily back propagation neural networks in the first subsystem to learn different categories of customer related information i.e. personal, social, campaign information etc. and the second subsystem utilised the predictions made from the first stage to make a final prediction. This method generated an AUC of 0.591

In [13], Neural Networks, Support Vector Machines, Classification and Regression Trees (CART) were utilised for classification. The research concluded with Neural Nets, SVM and CART having accuracies of 90.02%, 88.42% and 76.95% respectively.

2.5 Ensemble Methods

An ensemble of classifiers is a set of classifiers whose individual predictions are aggregated through weighted or unweighted voting to classify new examples. Ensemble methods have been a significant area of research and have been proven to usually perform much better than the individual classifiers they are constructed from [44]. A prerequisite for ensembles to boost the performance of their individual classifiers is to ensure the individual classifiers are independent and diverse [44]. Classifiers are said to be diverse is they make uncorrelated errors on new data instances [39]. Popular ensemble techniques include, bagging, boosting, stacking amongst a few others. In this sub-section, a high-level overview of two methods utilised in this research are presented.

2.5.1 Voting Classifier

This is a basic form of ensemble methods where the predictions of individual classifiers are aggregated and the final prediction is the class that obtains the most votes. This majority vote classifier is called a *hard voting* classifier.

When the individual classifiers have the capability to estimate class probabilities; ensembles with such constituents predict the highest class probability, averaged over all the individual classifiers. This is called *soft voting*. Soft voting usually performs better as it weighs highly confident votes more [39].

2.5.2 Adaptive Boosting

Boosting is any ensemble method which combines several weak earners to generate a stronger learner; usually, classifiers are trained sequentially, with each classifier trying to correct its predecessor's errors.

The Adaptive Boosting (AdaBoost) algorithm was developed by Freund and Schapire in [49]. It works by sequentially constructing various classifiers and then focusing the underlying algorithm on the training instances which were misclassified by the previous classifiers. Once all the constituents of the ensemble are trained, the AdaBoost classifier makes predictions by aggregating predictions from all the predictors through hard voting.

One major drawback of the AdaBoost algorithm is its sequential learning technique which means each individual classifier can only be trained following the training of the previous classifier; this implies the AdaBoost algorithm cannot be parallelized. Thus, an issue arises when the training is to be performed on large datasets as it can be computationally time consuming [39].

Table 2.1 presents a survey of literature which has undertaken classification on the bank marketing dataset using the AUC as the main evaluation metric. Other literature exists such as [13], [14], [23]. [46], [47] and [73] which performed classification on the bank marketing dataset, however these utilised accuracy as the main performance measure or just ROC curves. As will be further discussed in section 3.4.3, accuracy is an ineffective measure for comparison due to the unbalanced nature of the bank marketing dataset. Thus, these literatures were not considered for comparison.

Table 2.1: Survey of proposed methods evaluated on the Bank Marketing Classification problem using AUC as the performance Metric (All utilising the "Duration" attribute).

Author (Year)	**Method**	**Duration Attribute**	**AUC**
Moro, S., Laureano, R., & Cortez, P. (2012) [11]	Support Vector Machines	Included	0.938
J. Asare-Frempong and M. Jayabalan (2017) [43]	Random Forest	Included	0.927
Moro, S., Cortez, P., & Rita, P. (2014) [12]	Neural Network Ensemble	Included	0.794
Apampa, O. (2016). [48]	Random Forest	Included	0.766
S. Lahmiri (2017) [15]	Two-step Back – Propagation Neural Network	Included	0.5910

2.6 Model Validation and Selection

For a given classification problem, there are several algorithms which can be implemented to tackle the problem, each with different advantages and disadvantages. All these factors must be considered when selecting an algorithm to be utilised for a given problem. For this reason, a systematic approach is required to be able to choose the correct model from a shortlist of models, this is termed as model selection.

A basic pre-requisite for selecting a model will be that it generalizes well, i.e. the difference between its performance on the training set and unseen instances should be minimal. To validate a model's generalization ability, access to a dataset outside of the training dataset is required. One way to achieve this is by splitting the original dataset into three subsets; a training set, validation set and test set. However, if the original dataset is already small, this method may not prove efficient due to a reduced amount of training data and insufficient validation data.

A clever method to overcome this issue is through the use of *K-fold cross-validation*, this is a common technique applied in machine learning [33]. It involves randomly splitting the training set into K distinct subsets known as folds. The model is then trained on K-1 folds and evaluated on 1 fold and this is repeated K times. The validation set is then used to optimize regularization parameters, α and therefore prevent overfitting [9]. Following the selection of the optimal parameters on the validation set, the test set is then used to assess the final performance of a model.

2.7 Performance Measures for Classification

To evaluate a subset of algorithms, criteria known as performance measures are utilised for comparison. A Confusion matrix is a common method for visualising the results of a classifier. Several performance measures can be calculated from the variables of the confusion matrix shown in Table 2.2.

Where,

True Positive (TP) = samples *with* positive outcomes and predicted to have positive outcomes.

False Positive (FP) = samples *without* positive outcomes and predicted to have positive outcomes

True Negative (TN) = samples *with* negative outcomes and predicted to have negative outcomes.

False Negative (FN)= samples *without* negative outcomes and predicted to have negative outcomes.

Table 2.2: Generic confusion matrix for two classes

True Class	**Predicted Class**	
	Positive outcome	**Negative outcome**
Positive Outcome	True Positive (TP)	False Positive (FP)
Negative outcome	True Negative (TN)	False Negative (FN)

Table 2.3 presents the formulae for some of these performance measures which are derived from the confusion matrix. The selection of a performance measure to use is dependent on the characteristics of the training dataset and on the context of the problem being attempted.

Table 2.3: Performance measures derived from the confusion matrix

Performance Measure	Formula
Overall Accuracy	$\frac{TP + TN}{TP + FP + TN + FN}$
Sensitivity (Recall)	$\frac{TP}{TP + FN}$
Precision	$\frac{TP}{TP + FP}$
Specificity	$\frac{TN}{TN + FP}$
F1- Score	$\frac{2TP}{2TP + FP + FN}$

Common classification performance measures which have been employed in machine learning classification research such as [11], [12], [13] and [15] are briefly described below.

- Receiver Operating Characteristics (ROC) Curve: This is obtained by plotting the true positive rate against the false positive rate. It is used to visually analyse the performance of a model. A good performing classifier will ideally have a true positive rate of 1 and a false positive rate of 0. This implies the ROC curve of such a model will be very close to the upper left corner of the plot [9].

- Area Under the Curve (AUC): This is a quantitative measure of the ROC and is calculated by determining the area under the ROC curve. It ranges between 0 and 1; a good classifier will have an AUC value close to 1 [9].

- Mean Square Error (MSE): If $\hat{Y}$ is a vector of n predictions, and Y is the vector of the observed value of the variable being predicted, then the MSE is defined by the equation 2.19. A small value of MSE is a desired characteristic for models.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \quad (2.19)$$

3 DESIGN METHODOLOGY AND IMPLEMENTATION

This chapter describes the design methodology which was undertaken to meet the aims and objectives stipulated in section 1.1. The software applications, methods and techniques utilised are presented to provide transparency on the processes employed for obtaining the results of this research.

For each sub-section presented, the methodological process is outlined, followed by a discussion on the implementation of each process.

This research was implemented using the applications shown in the Table 3.1. The applications were selected based on the authors familiarity, ease of implementation and flexibility provided.

Table 3.1: Application Packages Utilized

Application Package	**Use**
Python	Core programming
Scikit-Learn	Machine Learning library
Keras	Neural Network library
Tensor Flow	Keras Backend
Plotly	Results visualization

3.1 Bank Marketing Dataset

As stated in section 2.2, this work utilised the bank marketing dataset from the UCI Machine Learning repository [1]. The dataset contains 41188 instances ordered from May 2008 to November 2010 with 20 input variables and one target variable. Table 3.2 describes each of the variables and the format they take.

3.1.1 Removal of the Duration Attribute

It has been established in [1], [11], [45] and [48] that the duration attribute significantly affects the outcome of a call to a client, i.e. the target variable. For instance, if duration = 0s, then y is ‘no’. In addition, it is a posterior attribute i.e. its value is not known prior to making a call. Therefore, the inclusion of this attribute leads to unrealistic models.

In line with the aims and objectives of this work, the duration attribute was excluded in order undertake a novel approach and implement realistic models. However, this method increased the difficulty of the classification problem due to the significance of the removed feature [45].

Table 3.2: Variables of the Bank Marketing Dataset

Variables	Variable Type	Description
Age	Numeric	Age of the client
Job	Categorical	Job type of the client (e.g. admin, blue-collar, entrepreneur etc.)
Marital	Categorical	Marital status of the client (e.g. married, single etc.)
Education	Categorical	Level of education achieved by the client (High school, Illiterate, University Degree etc.)
Default	Categorical	Has credit in default (Yes, No, Unknown)
Housing	Categorical	Has housing loan (Yes, No , Unknown)
Loan	Categorical	Has personal loan (Yes, No, Unknown)
Contact	Categorical	Type of contact (cellular, telephone)
Month	Categorical	Last month of contact by the bank
Day of Week	Categorical	Last day of contact by the bank
Campaign	Numeric	Number of contacts performed during the campaign and for the client instance
Pdays	Numeric	Number of days that passed since the client was last contacted from a previous campaign
Previous	Numeric	number of contacts performed before this campaign for the client instance
Poutcome	Categorical	outcome of the previous marketing campaign (failure, success, non-existent)
Emp.var.rate	Numeric	Portuguese employment variation rate -quarterly indicator
Cons.price.idx	Numeric	Portuguese consumer price index - monthly indicator
Cons.conf.idx	Numeric	Portuguese consumer confidence index - monthly indicator
Euribor3m	Numeric	Portuguese euribor 3 month rate - daily indicator
Nr.employed	Numeric	Number of employees - quarterly indicator
y	Binary	Output Variable - Has the client subscribed to a term deposit (Yes or No)

3.1.2 Data Splitting

The Bank marketing dataset was randomly split into 3 subsets using **Scikit-Learn: "*StratifiedShuffleSplit*"** class [66] which returns stratified random folds, where the ratio of sampling of each subset is similar to the original dataset to ensure a direct representation of the original dataset. The split ratio for datasets can be at the discretion of the author, however a common dataset split was utilised as follows:

- 1/3 of the dataset was used as the test set for measuring performance,
- The rest of the data set is split again into two sets, where 2/3 was utilised as the training set, and
- The remaining 1/3 was the validation set, for regularization.

After the implementation of the stratified shuffle split, the training set had a size of 20181 samples, the validation set, 8650 and the test set, 12357 samples adding up to the total of 41188 samples.

3.2 Data Pre-processing

As explained in Section 2.3, data pre-processing is important for obtaining optimal results. The following processes were employed as part of the pre-processing stage of this work.

3.2.1 Linear Separability

The linear separabilty of the bank marketing dataset was evaluated to help make an informed decision as to which algorithms would be adequate for the bank marketing classification problem. This was in conjunction with other factors as described in the aim and objectives in section 1.1.

The neural network (perceptron) was utilised for determining the linear separabilty of the dataset. As described in section 2.3.3 and 2.4.2 (Perceptron), if the dataset is linearly separable, the perceptron will only converge i.e. classify all instances correctly.

The **Scikit-learn:** ***"Perceptron"*** was utilised for implementing the perceptron model. Figure 3.1 show the confusion matrix obtained for the entire bank marketing dataset when predictions were made using the Perceptron algorithm. Analysing the confusion matrix in figure 3.1, the perceptron algorithm misclassified some instances which means the data is not entirely linearly separable. However, it correctly classified (35079 + 1440) out of a total of 41188 instances i.e. ~89% correctly. Based on this percentage, it can be concluded that the data is ~89% linearly separable, therefore linear models such as Logistic Regression or the Linear SVM can also be considered for classification.

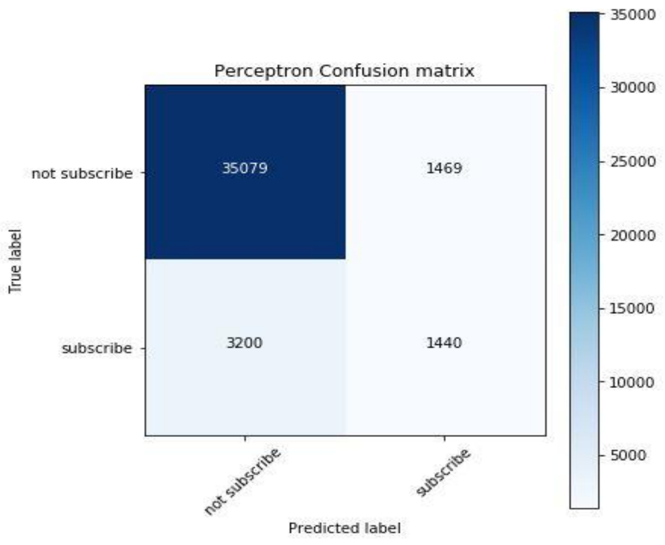

Figure 3. 1: The Perceptron confusion matrix on the bank marketing dataset showing ~89% linear separabilty

3.2.2 Up-sampling

Following a brief count analysis of the target classes, it was noted that the bank marketing dataset is imbalanced with approximately 88.7% of the samples assigned to the 'no' class and 11.3% to the 'yes' class, making the 'no' class the majority class with a ratio of approximately 8:1 ratio.

Imbalanced datasets can be detrimental to algorithms, especially those (Decision trees or Multilayer Perceptron etc.) which are designed to optimize accuracy without accounting for the relative distribution of each class [50]. Therefore, balancing the dataset is required to ensure each algorithm is training and evaluated equally [50]. Two common methods for handling imbalanced datasets include up-sampling and down-sampling. According to the research in [50], these two methods have similar efficacy. However, up-sampling was selected in this case as opposed to down sampling because of a significant reduction in the size of the training set when down sampling was applied.

Up-sampling was performed to match the number of samples in the majority class by resampling from the minority class. **Scikit-learn: '*resample*'** class [66] was utilised for implementing the up-sampling. Following this resampling, the ratio of the 'no' to 'yes' class became 1:1.

Up-sampling was only implemented on the training and validation sets, the final performance of the models was evaluated on the test set which was left unchanged.

3.2.3 Normalization and 1-of-n (one-hot) encoding

As shown in Table 3.2, the input variables contain two types of data; categorical data and numerical data. These two data types require different pre-processing techniques. The numeric data was *normalized* as described in section 2.3.2, whilst the categorical data was *one-hot encoded*.

For one hot encoding, each categorical attribute was mapped to a 1-of-n coding, thus increasing the number of attributes. For instance, for the 'Default' variable in Table 3.2 which has three classes ('yes', 'no' and 'unknown'), the first class was encoded as 1-0-0, the second class as 0-1-1 and the last as 0-0-1. This essentially created a binary attribute per class of the input variable. One hot encoding is required as efficient implementation of machine learning algorithms require numeric inputs [39].

Normalization and one-hot encoding were implemented using the **Scikit-learn:** ***"MinMaxScaler"*** and ***"OneHotEncoder"*** classes respectively [66]. The application of one-hot encoding increased the total number of features from 19 to 62.

3.2.3 Feature Selection

This stage involved selecting an optimal feature subset from the new total of 62 features (section 3.2.3). The Chi-square filter approach as described in Section 2.3.1 was selected for this aspect of pre-processing as opposed to the wrapper which is more computationally intensive. This method is advantageous for large datasets such as the bank marketing dataset, easy to implement and has been shown in [52] to generate relatively similar results to the wrapper approach.

Scikit-learn: '*SelectKBest*' class [66] was utilised for implementing the feature selection. The procedure implemented by this class was as follows:

- Selection of the best features by computing the chi-square statistic between each non-negative feature and class using equation 2.2 (section 2.3.1) and then,
- Removal of all but the *k* highest scoring features.
- *k* was initially selected as 62 and the class returned a score for each feature based on the chi-square calculation performed.

A *sequential forward search* (SFS) was used to search for the best feature subset starting from the highest scored feature and iteratively adding the next ranked feature until all features had been evaluated. The AUC performance measure was used to assess the performance of each feature subset.

3.3 Model Training & Regularization

The parametric classifiers (Logistic Regression and Naïve Bayes) explored in section 2.4.1 along with the non-parametric classifiers (Decision Trees, Linear Support Vector Machine and Multilayer Perceptron) in section 2.4.2 and the ensemble classifiers (Voting and Adaptive Boosting) in section 2.5 were evaluated on the bank marketing dataset.

This sub section outlines the methods for training and regularizing (if applicable) each of the classifiers listed above and how their implementation was performed in Python using the **Scikit-learn** and **Keras** libraries.

3.3.1 Gaussian Naïve Bayes

The training of the Gaussian Naïve Bayes (GNB) classifier follows on from equation 2.8 in section 2.4.1. A Maximum A Posterior (MAP) [9] estimation was used to estimate P(y) and $\mathrm{P}(x_i|y)$, where the former is the relative frequency of class 'y' in the training set.

The **Scikit-learn:** ***"GaussianNB"*** class [66] was used to implement the GNB algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$\mathrm{P}(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{\left(\frac{-(x_i-\mu_y)^2}{2\sigma_y^2}\right)} \tag{3.1}$$

where the parameters σ_y and μ_y were estimated using maximum likelihood [9]. No optimization was performed on the GNB model due to the highest conditional probability method which it uses to make predictions.

3.3.2 Logistic Regression

The objective of training the logistic regression model was to set a parameter vector, θ such that the model estimates high probabilities for positive instances (y=1) and low probabilities for negative instances (y=0). The Logistic regression cost function, known as the *log loss* [39] is shown in equation 3.2. This cost function was minimized using an optimization algorithm [39].

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y^i \log(\pi^i) + (1-y^i)\log(1-\pi^i)] \tag{3.2}$$

Where π is as per equation 2.7 (Section 2.4.1)

The Logistic Regression Model was implemented using the **Scikit-learn: "*LogisticRegression*"** class [66]. This class trains the algorithm using ***LIBLINEAR*** library [54] which was used to implement a Coordinate Descent optimization algorithm for minimizing the cost function.

Coordinate Descent Algorithm

Coordinate descent algorithms solve optimization problems by successively performing approximate minimization along coordinate directions or coordinate hyperplanes. Each step consists of evaluation of a single component of the gradient at the current point, followed by adjustment of the component of x, in the opposite direction to this gradient component.

The framework of the coordinate descent algorithm implemented by the ***LIBLINEAR*** library in **Scikit-learn** is outlined in Section 1.2 of [55].

Lasso Regression (*L1*) Regularisation

To avoid overfitting, the logistic regression model implemented was regularized [9] using the *Lasso Regression* which is short for Least Absolute Shrinkage and Selection Operator Regression (*Ridge regression was also implemented for regularizing the LR model, this method is discussed in section 3.3.4*).

The lasso regularization term takes the form shown in equation 3.3 [39]:

$$\alpha \sum_{i=0}^{n} |\theta_i| \tag{3.3}$$

This lasso regularization term is added to the logistic regression cost function shown in equation 3.4:

$$J(\theta) + \alpha \sum_{i=0}^{n} | \theta_i | \qquad (3.4)$$

This gives the final regularized minimization problem which was optimized by the coordinate descent algorithm. If a vector of the feature weights (θ_1 to θ_n), is defined as **w**, then the regularization term is equal to ||**w**||$_1$ where, ||·|| represents the $l_1$norm of the weight vector [39].

In **Scikit-learn**, the $\boldsymbol{l_1}$ norm was set by the penalty parameter ***'l1'*** of the ***"LogisticRegression"*** class whilst α was set by parameter ***C***, which is equal to $\boldsymbol{\frac{1}{\alpha}}$, thus reducing ***C*** increased the regularization of the model.

3.3.3 Decision Trees

As discussed in Section 2.4.2, the core objective of the Decision Tree algorithm is to find a split which minimizes the Gini Impurity measure in equation 2.18.

The Decision Tree algorithm was implemented using the **Scikit- learn:** ***"DecisionTreeClassifier"*** class [66]. This class used the *Classification and Regression Tree (CART)* algorithm to train the Decision trees. The objective of the *CART* algorithm was to minimize the cost function shown below:

$$\text{J(k,}t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right} \qquad (3.5)$$

Where $G_{left/right}$ measures the Gini Impurity of the left/right subset and,

$\frac{m_{left/roght}}{m}$ is the number of instances in the left/right subset.

The method for implementing the *CART* algorithm in **Scikit-learn (**adopted from [56]) is outlined in Algorithm 2.

ALGORITHM 2 – CART Algorithm

```
1. Step 1: Split training set, X in two subsets using a single feature k and a threshold $t_k$;
2. Step 2: Search for the pair (k, $t_k$) which produces the purest subsets;
3. repeat;
4. until Termination criteria is met;
```

Decision Tree Regularization Parameters

If left unconstrained, the Decision Tree algorithm is prone to overfitting due to its non-parametric nature [39]. The following parameters were utilised for regularizing the algorithm in **Scikit-learn** to avoid overfitting [9].

- ***"max_depth":*** This stipulates the maximum level of splits in the decision tree. E.g. in figure 2.5, the depth of the tree is 2 (the root node is at depth 0).
- ***"min_samples_leaf":*** This stipulates the minimum number of samples a node must have before it can be split [39].

3.3.4 Linear Support Vector Machine (Classifier)

As opposed to utilising Non-linear SVM methods such as Gaussian Radial Basis Function (RBF) Kernel or Polynomial Kernel [67], a Linear Support Vector Machine was applied due to its minimized computational complexity and thus significant reduction in computational time when training is performed on the processed dataset (section 3.2.3) which contains a high number of features.

The objective of training the Support Vector Classifier was to maximise the marginal distance between the target class variables as described in section 2.4.2. This objective is equivalent to the following optimization problem [58]:

$$\min_{\beta_o,\beta} \sum_{i=1}^{n} [1 - y_i(\beta_o + x_i^T \beta)]_+ + \frac{\lambda}{2} ||\beta||_2^2 \tag{3.6}$$

Where λ is a tuning parameter, same as β, and classification of input instances x is now given by $(\beta_o + x_i^T \beta)$. Equation 3.6 takes the form of a *loss* + *penalty* equation, similar to that in equation 3.4.

The loss function is of the format shown below, and is known as the *Hinge loss* [58].

$$(1 - yf)_+ \tag{3.7}$$

The maximizing margin property is intrinsic to the hinge loss function [59], therefore the *hinge loss* function was selected to train the Linear SVM model. This was implemented in **Scikit-learn** using the ***"LinearSVC"*** class [66].

The ***LIBLINEAR*** library [54] was also utilised for implementing the coordinate descent algorithm [55] for minimizing equation 3.6 i.e. the hinge loss function and the ridge regression term (discussed below).

Ridge Regression (*L2*) Regularization

Similar to the Logistic Regression Lasso regularization, the Linear SVM was also regularized to avoid overfitting [9]. In this case, *Ridge Regression* was utilised. The ridge regression is the penalty term in equation 3.6, where $||\cdot||_2$ is the $\boldsymbol{l_2}$ norm. The Ridge penalty shrinks the fitted coefficients to zero, thus, implementing regularization [58].

In **Scikit-learn**, the $\boldsymbol{l_2}$ norm was set by the penalty parameter ***'l2'*** whilst β was set by parameter ***C*** of the ***"LinearSVC"*** class. It must be noted that the l1 penalty term is not available to the ***"LinearSVC"*** class, therefore only ridge regularization (l2) was applied for the SVC model.

3.3.5 Multi-Layer Perceptron (MLP) Artificial Neural Network

A Multilayer perceptron was applied and evaluated on the bank marketing dataset. The method in which the algorithm was trained follows on from the training of the perceptron (which is the primary constituent of the MLP) in section 2.4.2. The step function outlined in equation 2.10 was updated to a function which has a well-defined non-zero derivate to enable the optimization algorithm for minimising the cost function to operate correctly [6], [39]. A common activation function used for MLP's is the sigmoid function [6], [39]. This function was utilised for training the MLP, its format is as follows:

$$\frac{1}{1+e^{-y}} \tag{3.8}$$

Due to the MLP's multiple output units, the cost function, $J(\boldsymbol{w})$ of the MLP was also amended from equation 2.12 in section 2.4.2; the amended cost function sums the costs over all of the networks output units. The amended cost function utilised is as follows:

$$J(\boldsymbol{w}) \equiv \frac{1}{2}\sum_{x\in X}\sum_{k\in outputs}(y_{kx}-\hat{y}_{kx})^2 \tag{3.9}$$

where *outputs* is the set of outputs units in the network, and y_{kx} and $\hat{\mathrm{y}}_{\mathrm{kx}}$ are the target and output values of the kth output unit and training example x.

The *Adam* optimizer [60] described in Section 2.4.2 was utilised for minimizing the MLP cost function in equation 3.9. Along with the advantages of the Adam optimizer highlighted in section 2.42, the Adam optimizer was chosen as it has been shown to scale well for large-scale machine learning problems with large datasets and dimensionality [60].

Training the MLP then involved incorporating the Adam optimization algorithm with a differentiation technique used to compute the *gradient* parameter (see line 8b of *Algorithm 1* in section 2.4.2) required in the *Adam* algorithm. The differentiation technique utilised was the *backpropagation* algorithm [6]. The backpropagation algorithm for feedforward networks can be summarised by Algorithm 3, adopted from [6] and [61].

ALGORITHM 3 – Backpropagation Algorithm

```
1. Create a feedforward network with X inputs, H hidden units, and K output units.
2. Initialize all the network weights to a small random values
3. **For** each training instance in X **do**
4. Forward propagate the input through the network:
5. Input the instance $x$ to the network and compute the output of every unit in the network
6. Back propagate the errors through the network:
7. For each network output unit $Y$, calculate its error term $\delta_k$
       $\delta_k \leftarrow \hat{y}_k(1-\hat{y}_k)(y_k-\hat{y}_k)$
8. For each hidden unit h, calculate its error term $\delta_h$
       $\delta_h \leftarrow \hat{y}_h(1-\hat{y}_h)\sum_{k\in outputs} w_{kh}\delta_k$
9. Update each network weight $w_{ji}$
       $w_{ji} \leftarrow w_{ji} + \eta\delta_j x_{ji}$
10. repeat;
11. **until** Termination condition is met;
```

An in-depth explanation of the backpropagation algorithm can be found in section 4.5.2 in [6] and section 6.2 in [61].

Keras [74] neural network library was utilised for implementing the MLP, a ***"Sequential"*** model was implemented where the all the parameters of the MLP such as the number of inputs, hidden neurons and outputs, the activation function, type of optimizer and regularization parameters (discussed below) utilised were specified.

MLP Regularization

The neural network can be prone to overfitting as it possesses a high degree of freedom and is allowed to fit complex functions including the noise associated with these functions. These noisy functions do not depict the true characteristics of the training sample and will thereby lead the neural net to perform poorly on unseen data. The techniques implemented in **Keras** for regularizing the neural network are described below.

- ***L2 Regularization (Weight Decay)***

To counter overfitting, l2 regularization (weight decay) was implemented, this is similar to the Ridge Regression regularization [58] used for SVM's. The weight decay parameter penalized the large weights of the neural network layers during training, therefore reducing their ability to learn complex decision surfaces and thus, enabling better generalisation performance [62].

Weight decay was incorporated into the cost function (equation 3.9) algorithm as follows:

$$J(\boldsymbol{w}) + \tfrac{1}{2}\lambda\textstyle\sum_i {w^2}_i \qquad (3.10)$$

Where λ, is the penalty term that penalizes large weights.

The weight decay was implemented using the ***"kernel-regularizer"*** method (l2(λ)) as part of the ***"Sequential"*** model in **Keras** [74].

- ***Early Stopping***

Early stopping simply stops the training of the MLP when the validation error stops improving, thus, minimizing overfitting.

Early stopping was also applied as part of the ***"Sequential"*** model in **Keras** [74]**.** A "patience" parameter was utilised to stipulate the number of training iterations for which the validation error should be observed for non-improvement before stopping the MLP training.

- ***Dropout***

Dropout [63] was used for regularizing the MLP. This method temporarily drops out units (excluding the output units) from the network at every training step, where each unit has a fixed probability *p* of being dropped out.

Dropout was implemented for the input units as part of the ***"Sequential"*** model in **Keras** [74] whilst training the MLP. The probability p was selected using cross validation.

3.3.6 Adaptive Boosting

The implementation of the AdaBoost algorithm was carried out using the **Scikit-learn: "*AdaBoostClassifier*"** class [66]. The implementation of this algorithm is based on the multiclass version of the AdaBoost known as *SAMME* (Stagewise Additive Modelling using a Multiclass Exponential loss function), this is equivalent to the AdaBoost when there are just two classes [39]. The multiclass AdaBoost algorithm which was implemented is outlined in section 1.1 of [64]. The procedure is as follows

- Starting with the unweighted training sample, the AdaBoost builds a classifier that produces class labels.
- If a training data point is misclassified, the weight of that training data point is increased (boosted).
- A second classifier is built using the new weights, which are no longer equal.
- Again, misclassified training data have their weights boosted and the procedure is repeated.

The parameters utilised for optimizing the algorithms were the learning rate parameter and the number of classifiers in the boosting sequence applied.

Due to the computational time required for training the AdaBoost algorithm, a Decision Tree with a max depth of 1 was selected as the base classifier for the AdaBoost. Using these parameters reduced the time for training and was sufficient to meet the aims of this project i.e. evaluating the improvement derived from ensemble methods.

3.3.7 Voting Classifier

All the classifiers (excluding the MLP) discussed previously were utilised for implementing a Voting Classifier. Following the training of individual classifiers on their respective feature subsets, they were used to make predictions. The predictions obtained were aggregated and a majority voting system was used to determine the output of the final classifier. The MLP could not be implemented due to compatibility issues between ***Scikit-learn*** and ***Keras***. *Hard voting* as opposed to *soft voting* (see section 2.5.1) was utilised because the **Scikit-learn:** ***"LinearSVC"*** class did not have the capability to estimate class probabilities.

The voting classifier method was implemented using the **Scikit-learn:** ***"VotingClassfier"*** class. This method performs the majority voting system as discussed in Section 2.5.1.

3.4 Model Validation and Evaluation

The optimization, generalization assessment and overall performance evaluation of the investigated models involved utilizing the methods outlined below. These methods were implemented using the **Scikit-learn** library and **Plotly** visualization tool.

3.4.1 Learning curves

A learning curve is a plot of the generalization performance against the amount of training data [65]. This analytical tool was used to assess if the models which were investigated demonstrated signs of *overfitting, under fitting* or both. The insights generated from empirically evaluating a models performance using the learning curve enabled the selection of optimal regularization hyper parameters.

Learning curves were implemented using the **Scikit-learn:** ***"Learning_curve"*** class [66]. 7-fold cross validation (Section 2.6) was performed on varying subsets of the initial training set, then the generated cross-validation scores were averaged over the 7 runs. These cross validation scores which were optimized based on the *MSE* measure were then used to plot the generalization performance against the training set size.

A learning curve was plotted for each algorithm implemented, and for the algorithms which were regularized, a learning curve was plotted post-regularization to observe the impact of the regularization implemented.

3.4.2 Grid Search Cross-Validation

The appropriate regularization hyper-parameters for each model discussed in section 3.3 were selected using *grid-search* method with 5-fold cross validation. Different pairs of regularization parameters (such as minimum samples leaf and max depth for the Decision Tree) were tested and the pairs with the best cross-validation score was chosen. Amongst the several methods for selecting parameters, the grid-search method was chosen because it conducts an exhaustive parameter search whilst still having similar computational time to other advanced methods such as *cross-validation rate approximation* [67]. For some of the models investigated, a wide range parameter search was conducted initially to narrow down the parameter search space, then a parameter search with increased resolution (fine) was carried out to find the optimal parameters. These optimal parameters were then evaluated on the validation set to obtain an independent performance

The grid-search method was implemented using the **Scikit-learn:** ***"GridSearchCV"*** class [66].

Contour plots were generated using **Plotly** [75] to help visualize the results obtained from the ***"GridSearchCV"*** class. The contour plots show the variation of the *negative Mean Squared Error* displayed on the right of the plot (with "red" signifying a desired value) against a pair of regularization parameters (x & y axis).

Incorporating the results with **Plotly** was relatively straightforward for most models. However, for the MLP model implemented through **Keras** [74], a special program was hard-coded to *de-serialize* the grid search results generated from the *Keras wrapper* and then incorporate this with the **Plotly** [75] contour plot class.

3.4.3 Performance Measures and Testing

As stated in section 3.1, the bank marketing dataset is imbalanced; He et al in [68] stipulate that conventional methods for evaluating algorithms using singular assessment criteria such as accuracy do not provide adequate information for imbalanced datasets. It is therefore ineffective for measuring the performance of classifiers as it is difficult to make relative analysis due to the sensitivity of the accuracy metric to class distributions. The ineffectiveness of accuracy for imbalanced datasets is also concluded by Guo et al in [69] and Maloof in [70].

He et al in [68] suggest more informative assessment metrics, such as the receiver operating characteristics curves, precision-recall curves, are necessary for conclusive evaluations of performance in the presence of imbalanced data.

For this paper, the main performance measure used for assessing the performance of each algorithm was the area under the ROC curve (***AUC***) (see section 2.7). This method presents the advantage of being independent of class frequency [12] and provides a good summary from comparing classifier across different conditions [25]. Recall was also assessed to gain insights on the amount of correct predictions of the 'subscribed'/'yes' class.

Following the validation of all the parameters and features subsets, the optimal combinations for an individual classifier was used to train the final classifier and then the final performance metric on the test set was calculated using the chosen feature subset.

4 RESULTS AND DISCUSSION

In this section, the results obtained during the training and the optimization of the seven classifiers implemented are presented. The results include learning curves for assessing the generalization of the models, grid search parameter optimization results, ensemble methods result and a comparison of the overall perform of the classifiers on the test set. The findings from the results are then critically discussed to assess if the aims and objectives of the project were met.

During the training and optimization phase of the project, a random seed was utilised to ensure the data set splits (section 3.1.2) were constant and reproducible. However, during the final test phase, this seed was removed and 10 test runs were conducted to assess the consistency of the results obtained.

4.1 Results

Each classifier was trained, optimised and tested individually. The results for the feature selection and the results obtained for each individual algorithm is presented in this sub-section.

4.1.1 Chi-Square Feature Selection

The chi-square feature selection process was implemented as discussed in section 3.2.3. Table 4.1 and 4.2 presents the 62 *normalized* features of the bank marketing dataset in descending order of their chi-square scores. It is observed from table 4.1 that the top 5 attributes which significantly impact the classification outcome are the *previous campaign outcome, euribor 3-month rate, contact via telephone, Portuguese employment variation rate* and the *number of employees*. These results are somewhat intuitive as the outcome of the previous campaign will most likely reflect the outcome of a current campaign. In addition, economic factors such as the employment and interest rates can be equated to client's willingness to spend, and therefore subscribe to buying a financial product.

Due to the sequential forward search method described in section 3.2.3, from hereon in, the number of features (denoted as '*nof')* selected for the feature subset of an algorithm instance would equate to the first *nof* features listed in Tables 4.1 and 4.2.

4.1.2 Gaussian Naïve Bayes

The Gaussian Naïve Bayes (GNB) model was implemented as discussed in section 3.3.1 whilst incorporating the feature selection process discussed in section 3.2.3. The results of the sequential forward search (SFS) for the GNB model is shown in Figure 4.1. This figure illustrates that the best AUC score for the GNB is obtained when a feature subset containing the first 8 features from table 4.1 is used. It is also observed that the addition of the last 32 features from tables 4.1 and 4.2 to the training feature subset does not seem to improve the AUC score for the GNB model. Using the first 8 features from table 4.1 for training the GNB model, the learning curve obtained after training is illustrated in Figure 4.2.

Looking at the training error in figure 4.2 for approximately one training sample, the GNB model has an initial training error of about 0.31, it is unable to perfectly fit one training sample, implying the model is highly biased at this instance i.e. under fitting (section 2.4). The error increases sharply as the training set size increases, and then starts to decrease as the training set size increases further. The error eventually starts to settle at about 16000 training set samples.

At one training sample, the validation error is significantly greater than the training error. This implies that the GNB model has poor generalization performance for small training set sizes. However, as the training set size increases, the validation error starts to decrease at a similar rate to the training error, eventually converging at a similar error as the training error (remaining higher than the training error averaged over the full training sample). This behaviour shows that as the training size increases, the GNB model begins to learn from the data, leading to an improvement in its generalization performance. From 16000 training samples onwards, both errors remain relatively stable at approximately 0.3, showing that the model is no longer learning from the additional data and has reached its maximum performance.

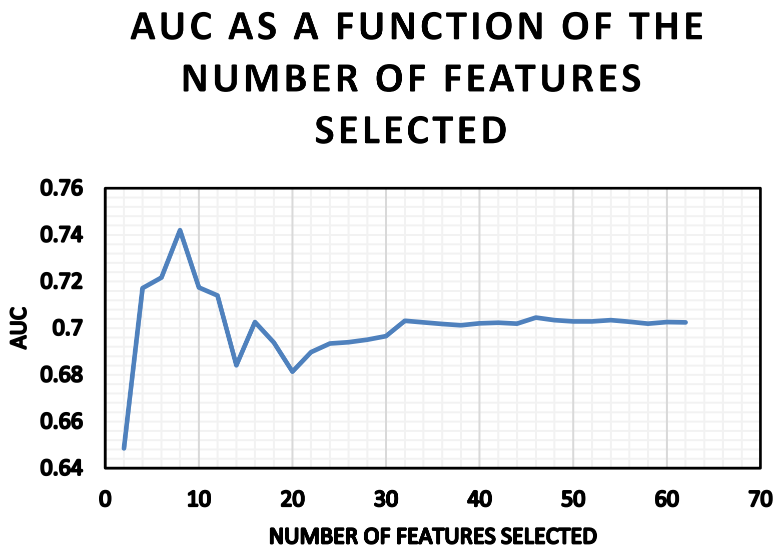

Figure 4. 1: AUC as a function of the number of features selected for the Gaussian Naive Bayes Model showing the best AUC score is obtained with a feature subset of the top 8 features in table 4.1.

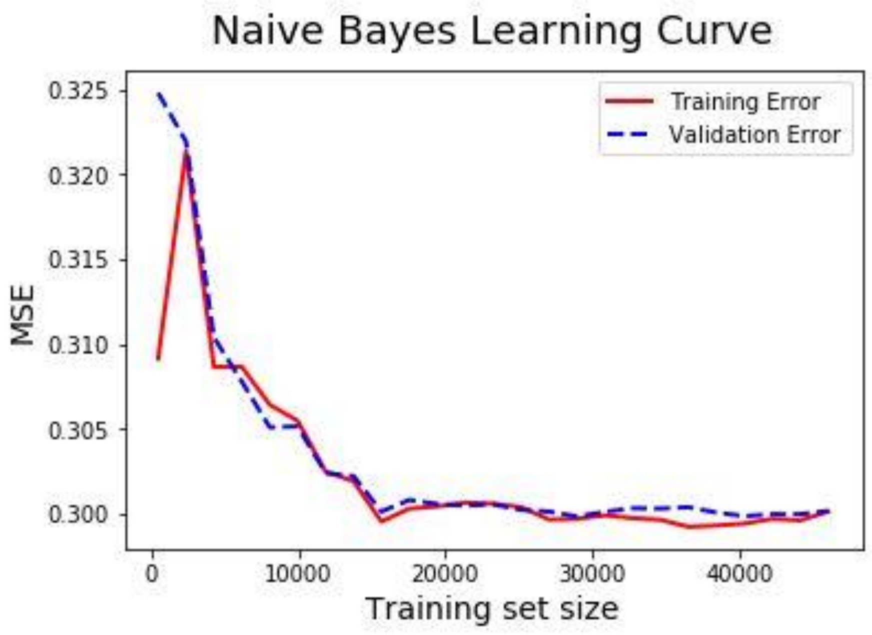

Figure 4. 2: MSE as a function of training set size for the GNB. This model initially demonstrates poor generalization performance and doesn't fit the training data perfectly for very small training set sizes. As the training set size then increases, the models generalization performance increases and the training and validation MSE decreases until a certain point where the model stops learning from new data.

Table 4. 1: Normalized feature list for the bank marketing dataset sorted in descending order of Chi-square score (Part 1)

Normalized Feature Name	**Chi-Square Score**
poutcome_success	2821.371798
euribor3m	2535.97971
contact_telephone	1560.891218
emp.var.rate	1430.78058
nr.employed	1335.262041
month_may	840.095042
default_unknown	786.048734
month_oct	721.985965
month_sep	706.067658
month_mar	681.57717
contact_cellular	609.508172
previous	527.267036
poutcome_nonexistent	480.527001
job_blue-collar	476.778576
job_retired	460.848417
month_apr	401.808027
pdays	371.70078
job_student	336.641439
education_basic.9y	216.700141
month_dec	197.415367
marital_single	184.983397
cons.price.idx	162.787932
default_no	151.68994
education_university.degree	147.490517
job_services	122.533507
marital_married	86.265495
month_jul	84.944803
job_admin.	70.325912
poutcome_failure	48.034129
job_unemployed	33.95391
campaign	32.609251

Table 4. 2: Normalized feature list for the bank marketing dataset sorted in descending order of Chi-square score (Part 2)

cons.conf.idx	30.990954
day_of_week_mon	29.946286
month_nov	25.962055
education_basic.6y	22.161701
month_aug	21.577668
day_of_week_tue	21.147475
education_unknown	20.714419
housing_yes	12.137918
housing_no	10.557865
day_of_week_fri	8.967559
education_basic.4y	8.708871
education_high.school	6.99282
age	6.734453
housing_unknown	6.396544
loan_unknown	6.396544
job_housemaid	4.343474
day_of_week_wed	4.031907
job_technician	3.564479
job_entrepreneur	3.03131
day_of_week_thu	2.510242
marital_divorced	2.469135
education_professional.course	1.694369
loan_yes	1.653799
job_self-employed	1.190032
loan_no	0.917103
education_illiterate	0.884873
marital_unknown	0.389658
month_jun	0.291446
job_unknown	0.127728
job_management	0.037389
default_yes	0

4.1.3 Logistic Regression

The Logistic Regression (LR) model was initially implemented using an initial parameter of C=1 (this is the **Scikit-learn** default value for C, see section 3.3.2) and with an incorporation of the chi-square feature selection process. The results of the sequential forward search conducted for the Logistic regression model is shown in figure 4.3. From this figure, it is observed that the optimal number of features which gives the best AUC score is either 44, 45 or 46. Since all 3 feature subsets have the same AUC score, the feature subset with 46 features was chosen arbitrarily. The figure also shows that adding the last 12 features from table 4.2 have no effect on improving the AUC performance. A learning curve was plotted to observe the generalization performance of the logistic regression model with C=1, this is displayed in figure 4.4.

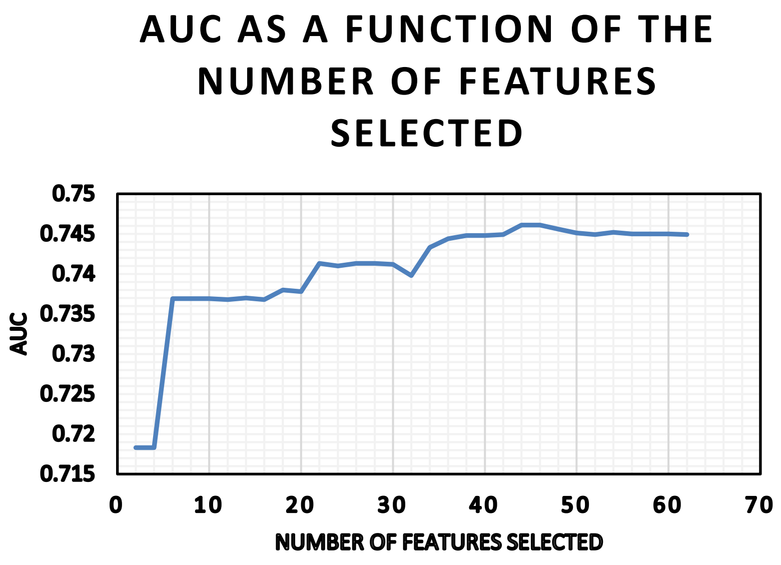

Figure 4. 3: AUC as a function of the number of features selected for the logistic regression model showing the best AUC score is obtained with a feature subset of the top 46 features in tables 4.1 and 4.2

Observing the training error in figure 4.4 which begins at zero (the MSE axis is restricted to the range shown to allow more precision for reading most MSE values), the model is overfitting the training data for very small training set size e.g. 1 sample because the models complexity significantly outweighs the data size. On the other hand; for a very small training set size of 1 sample, the validation error is very high relative to the training error. This implies the model has poor generalization performance at the point.

As the training set size begins to increase, the training and validation start to converge, signifying an improving generalization performance. The training error increases then decreases as the model begins to learn for the training data. However, a point is reached (~19000 samples) when the model stops learning from additional data and both the training and validation error plateau. This is a sign of high bias (under fitting).

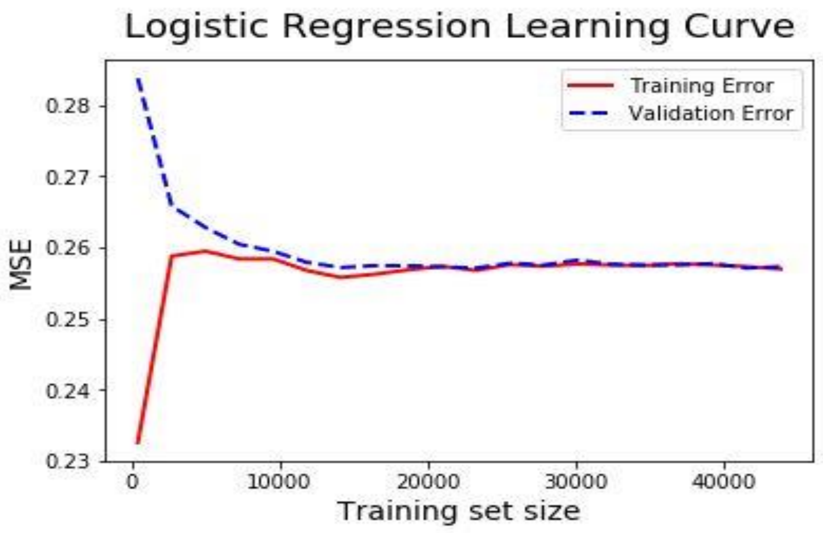

Figure 4. 4: MSE as a function of the training set size for the Logistic Regression model. This figure shows for small training set sizes, the LR model has poor generalization performance and is overfitting the training data. As the training set increases the generalization performance increases but both the training and validation error plateau, the model is no longer learning from additional data.

After the initial training and feature selection, a wide range and fine grid search (see section 3.4.2) were conducted to further optimize the LR model using the regularization parameters: C, l1 and l2 penalty terms (see section 3.3.2 and 3.3.4) and the 46 selected features. The wide range search was between C=0.1 and C=50. The grid-search contour plots are presented in figures 4.5 and 4.6. The parameters are optimized with the AUC score which is displayed on the right of the plots. The red colour signifies the desired parameter region i.e. high AUC score. The wide range search in figure 4.5 shows that the optimal parameter region is with the l1 penalty and C<5. The fine search was then conducted with these parameters, figure 4.6 shows the best AUC score was obtained with **C=0.5** and penalty **l1**.

Following the optimization of the LR model using the grid-search method. An updated learning curve was plotted to observe effect of the optimization. Figure 4.7 shows this updated learning curve.

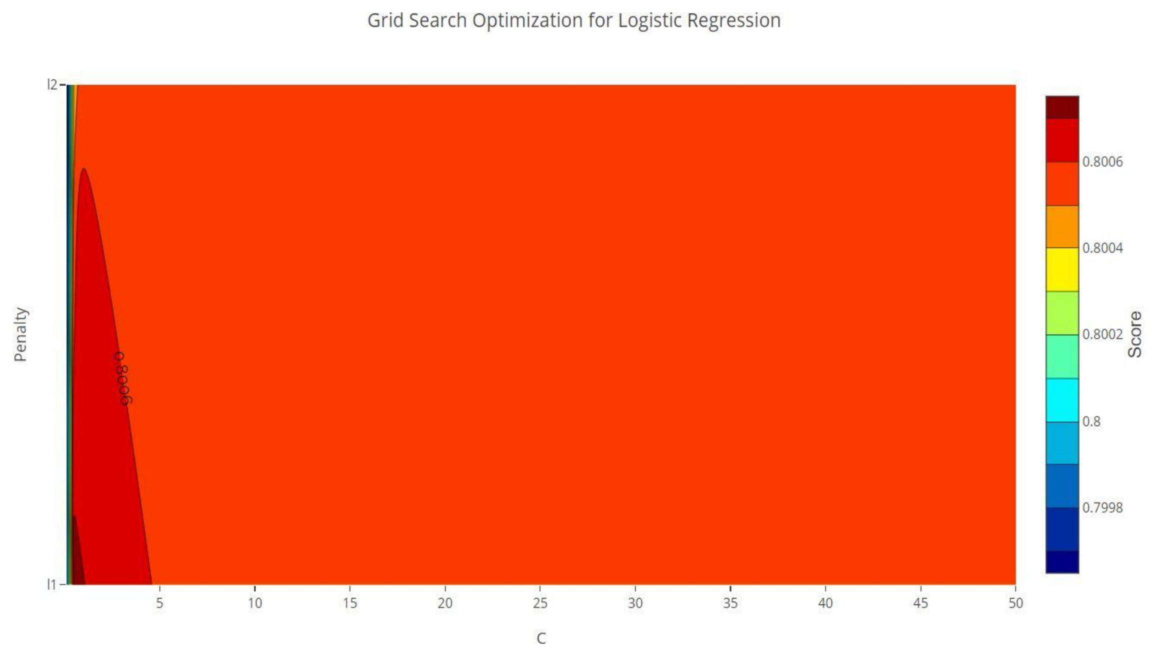

Figure 4. 5: Wide range grid-search contour plot for regularization parameter C against lambda (section 3.3.2) where the "Score" is the AUC score being optimized. On the colour bar to the right of the plot, red colour depicts the best AUC score.

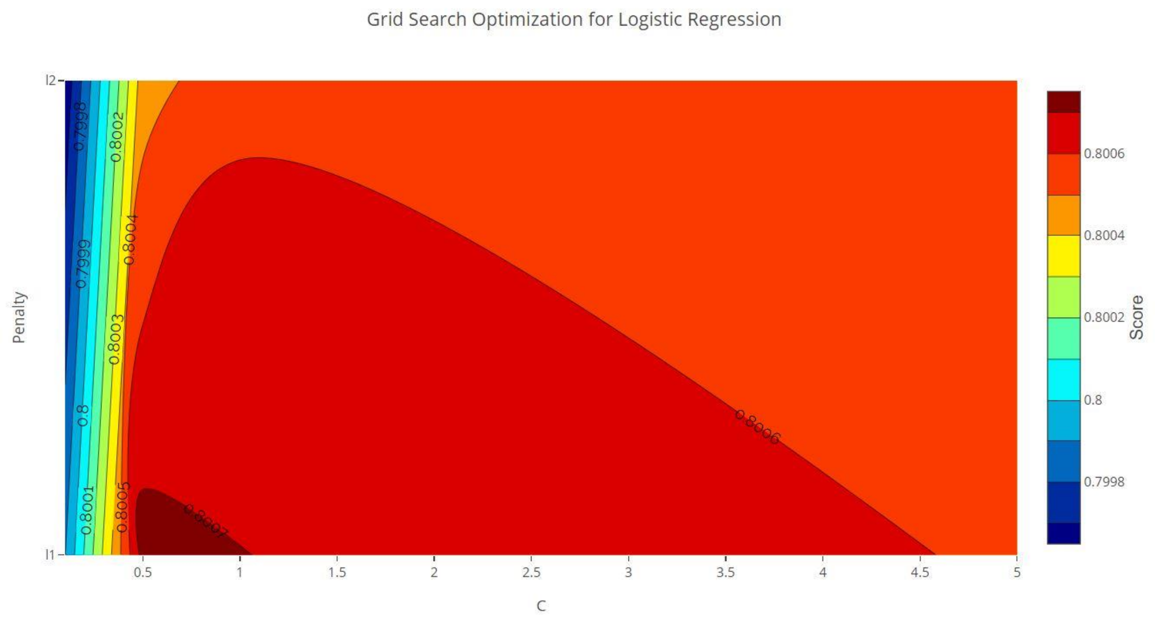

Figure 4. 6: Fine grid-search contour plot for regularization parameter C against lambda (section 3.3.2) where the "Score" is the AUC score being optimized. On the colour bar to the right of the plot, red colour depicts the best AUC score

Comparing the LR model in figure 4.4 with the optimised model in figure 4.7, the same conclusions drawn. However, for the optimised case, the MSE error converges at a lower value ~ 0.2557 which is an improvement of ~2% when compared to the MSE of ~0.26 for the initial LR model.

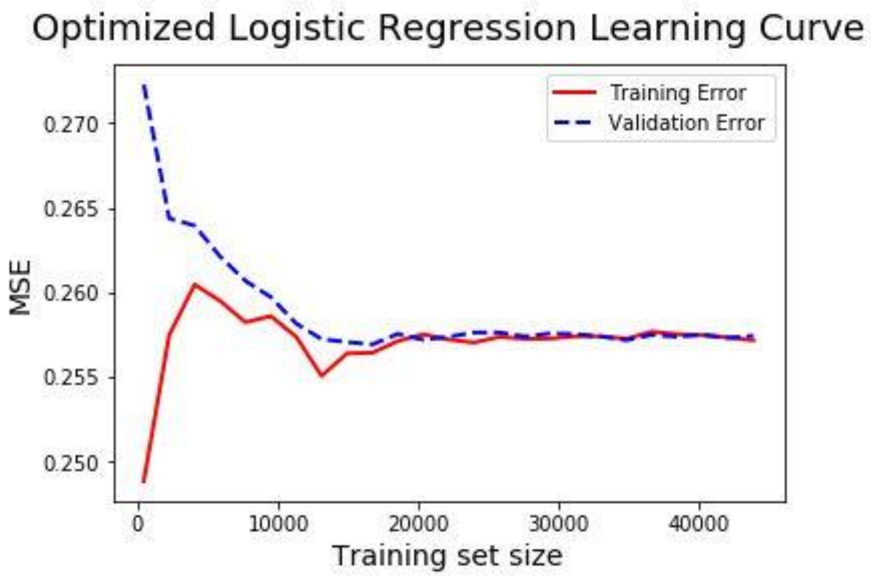

Figure 4. 7: MSE as a function of the training set size for the optimized Logistic Regression model. This figure follows a similar trend to figure 4.4. However, for the optimised case, the MSE error converges at a lower value ~ 0.2557 compared to ~0.26 for the initial LR model

4.1.4 Artificial Neural Network

The initial architecture of the Multilayer Perceptron Artificial Neural Network (MLP-ANN) trained was designed as described in section 2.4.2 and 3.3.5. 62 neurons were used in the input layer, 1 neuron in the output layer and 1 hidden layer.

A learning curve was not implemented for the neural network, due to a lack of compatibility between the implementation of the MLP using **Keras** and the implementation of the learning curve class using **Scikit-learn**. Instead, to analyse the generalization performance of the network, a graph of the MSE as a function of the number of training iterations (epochs) [9] was plotted for varying number of neurons in the hidden layer. Figures 4.8, 4.9 and 4.10 show the graphs obtained for 1,5 and 10 hidden neurons respectively. The number of epochs (40) used was sufficient enough to make an inference on the generalisation performance for each number of hidden neurons without having to train each ANN-MLP for a prolonged time period.

At the onset of training in figure 4.8, the training error is high. However, once the 1st training iteration passes, the training error converges within ~7 epochs and reaches its minimum MSE. This graph suggests that the MLP is highly biased (under fitting) at the onset of training, but quickly learns from the data after the 1st iteration after which no further learning occurs. The validation error also reduces sharply and then converges to its minimum value which is higher than the training error in approximately 1 iteration. After the 1st iteration of training, the generalisation performance remains constant with no improvement as the number of epochs increases.

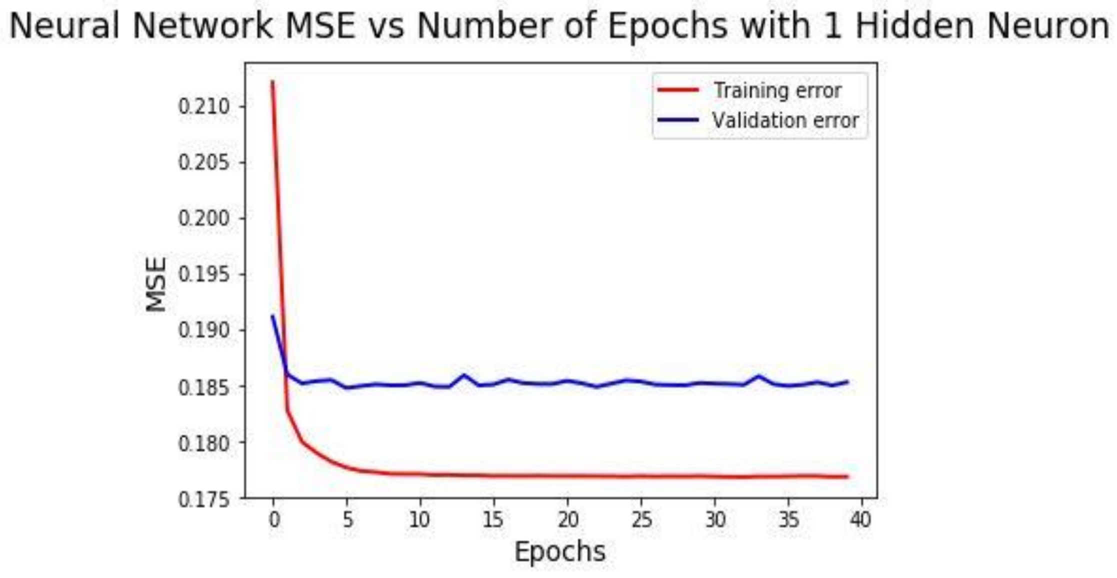

Figure 4. 8: MSE as a function of the number of epochs for a neural network with 1 hidden neuron. This network rapidly reaches its optimum performance after ~7 training epochs with the validation error remaining stable on average at about 0.185.

In figures 4.9 and 4.10, the training error decreases sharply after the 1st iteration and continues to decrease as the number of epochs increase. The downwards trajectory of the training error implies the error will continue to decrease until convergence if the training carried on for 40 + n number of epochs. Looking at the validation errors in both figures, there is an upwards trajectory for the error from approximately 10 training epochs. The trajectories of the training and validation error as the number of epochs increases suggest both MLP’s are overfitting, thus leading to increasingly poor generalisation performance.

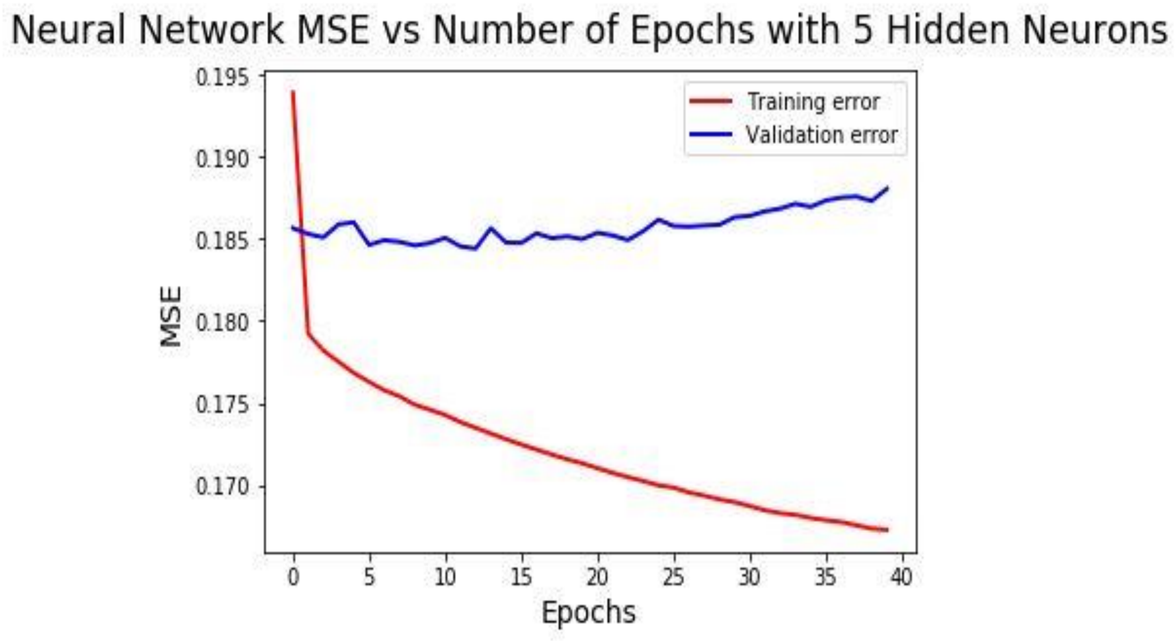

Figure 4. 9: MSE as a function of the number of epochs for a neural network with 5 hidden neurons. This figure shows the overfitting phenomena as the training and validation errors diverge from each other as the number of training epochs increases past 10 epochs.

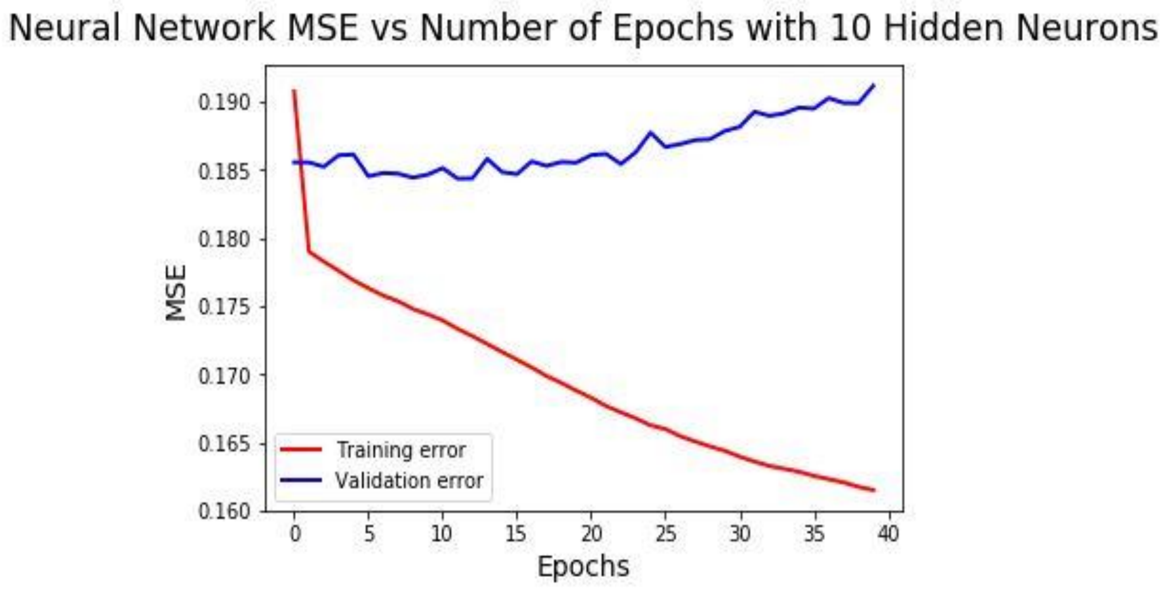

Figure 4. 10: MSE as a function of the number of epochs for a neural network with 10 hidden neurons. This figure shows overfitting phenomena as the training and validation errors diverge from each other as the number of training epochs increases past 10 epochs.

Following the generalization performance assessment, the weight decay (lambda) parameter was added to the MLP for regularization as described in section 3.3.5. A 5 - fold grid-search cross-validation was then implemented to select the optimal parameter pair of *lambda* and *number of hidden neurons*. A wide range (1≤ hidden neurons≤ 40 & 0.0001 ≤ lambda ≤ 0.5) grid search was initially conducted followed by a fine grid search. The results for both searches is presented in figures 4.11 and 4.12 respectively. The grid-search was optimized against the negative mean squared error with the red regions depicting the best negative MSE score.

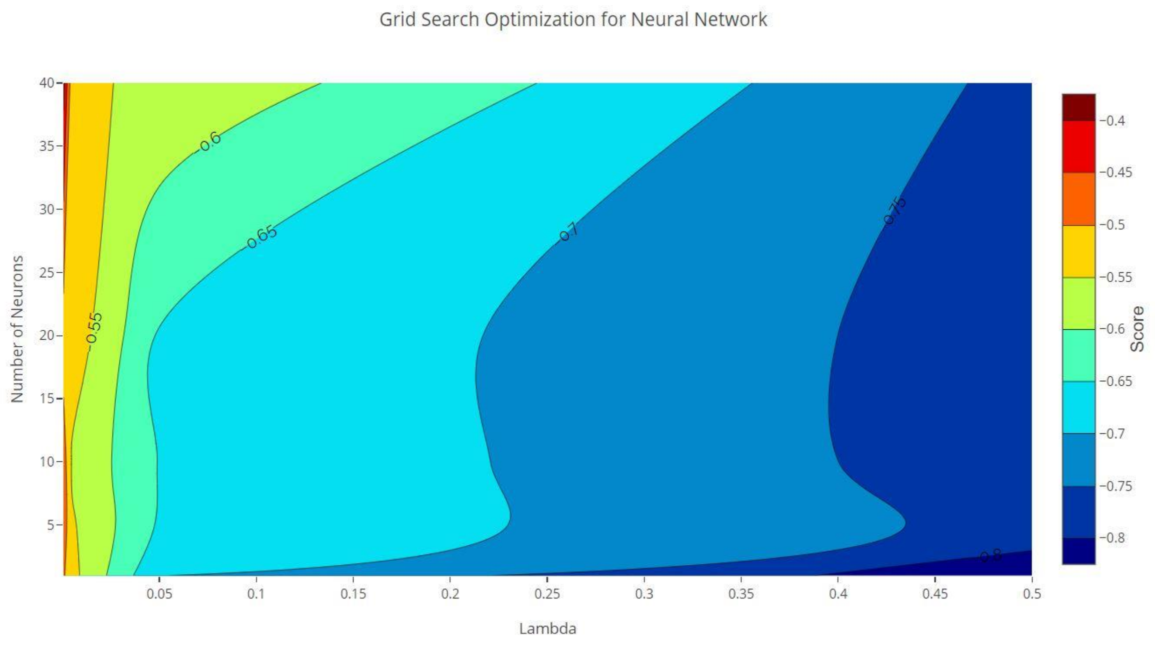

Figure 4. 11: Wide range grid-search contour plot for number of hidden neurons against lambda where the "Score" is the negative MSE being optimized. On the colour bar, red depicts a desired score. Thus, a fine grid-search should be performed around very small lambda and between 1 and 10 hidden neurons

Figure 4.11 shows that the optimal region for the parameter pair is located at low lambda values (< 0.005) and between 1 and 10 hidden neurons. Using the finer grid-search as shown in figure 4.12, the optimal parameter pair becomes clear i.e. *5 neurons and 0.0005*.

Utilising the following parameters for the initial MLP architecture: 62 input units, 5 hidden, 1 output unit and 0.0005 lambda, the chi-square feature selection was applied to determine the optimal number of features for the MLP.

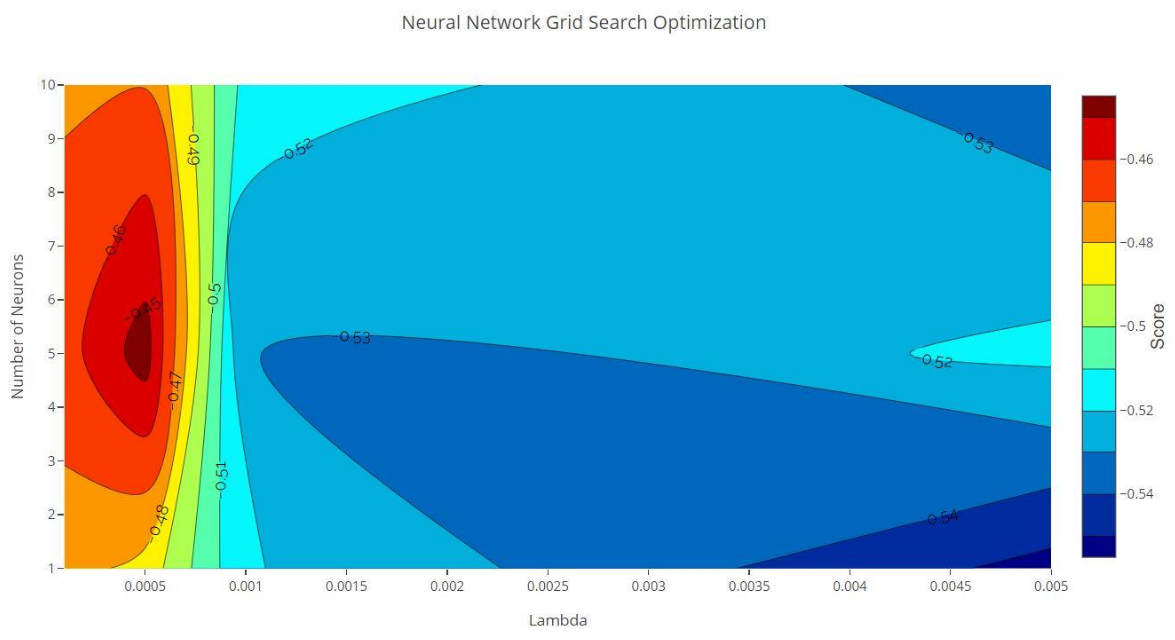

Figure 4. 12: Fine grid-search contour plot for number of hidden neurons against lambda where the "Score" is the negative MSE being optimized. On the colour bar, red depicts a desired score. Thus, the optimal parameters are obtained at 5 hidden neurons and 0.0005 lambda

Figure 4.13 shows the results obtained from the sequential forward search for the MLP. Upon the analysis of this figure, the number of features giving the best AUC score for the MLP was seen to be 52 features. With this optimal number of features, a further grid-search was conducted to determine the final parameters of the MLP, the contour plot in figure 4.14 illustrates a parameter pair of **10 hidden neurons and lambda = 0.0003** were optimal for the 52 feature subset.

The generalization performance was re-assessed to analyse the impact of the parameter optimization and regularization applied on the MLP. Figure 4.15 shows the updated MSE as a function of the number of epochs graph. Upon the inspection of figure 4.15, both the training and validation errors as seen to be decreasing as the number of epochs increases. This is an improved performance in contrast with the performance depicted in figures 4.9 and 4.10 as the model is no longer showing signs of overfitting. There is now a downwards trajectory for the validation error after 10 epochs up until 50. However, there is still room for improvement as there exists an error gap of ~6% between the training and validation curves.

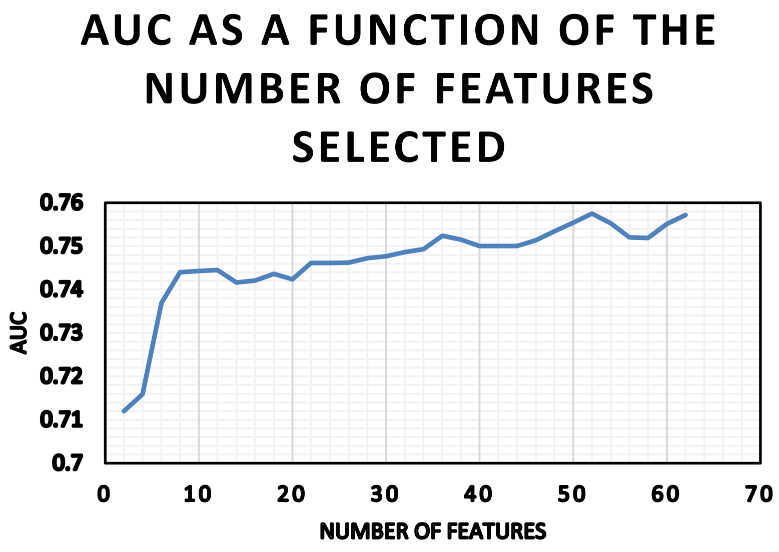

Figure 4. 13: AUC as a function of the number of features selected for the MLP showing the optimal number of features at 52

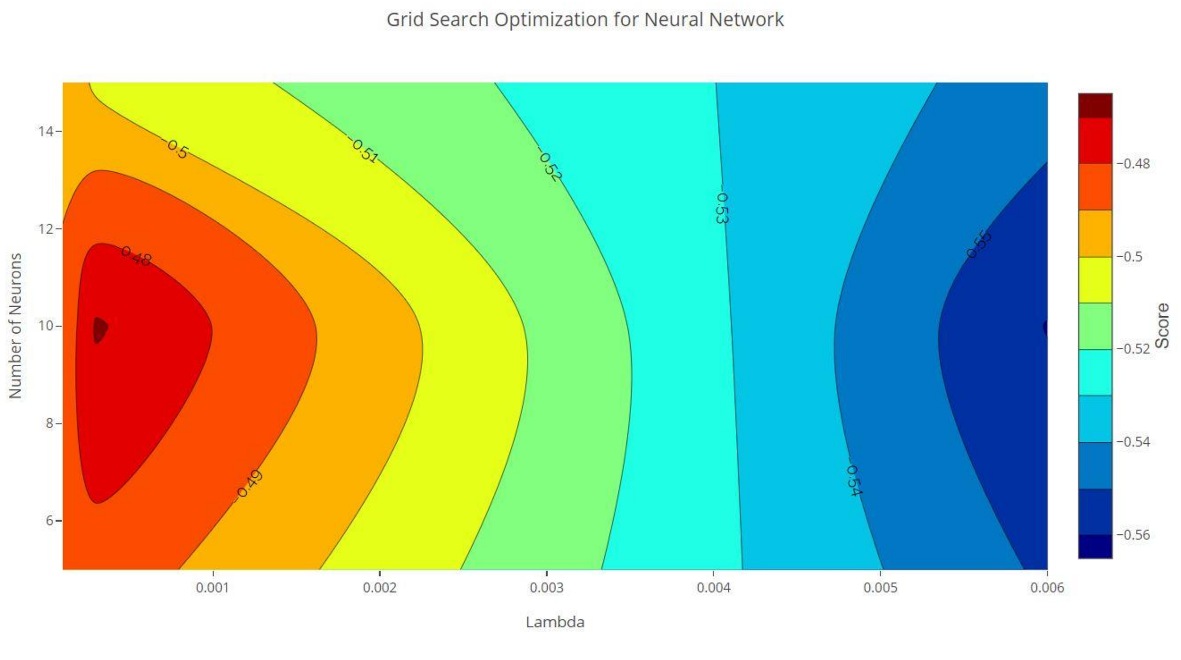

Figure 4. 14: Fine grid-search contour plot for number of hidden neurons against lambda after chi-square feature selection where the "Score" is the negative MSE being optimized. On the colour bar, red depicts a desired score. Optimal parameters obtained are 10 hidden neurons and 0.0003 lambda.

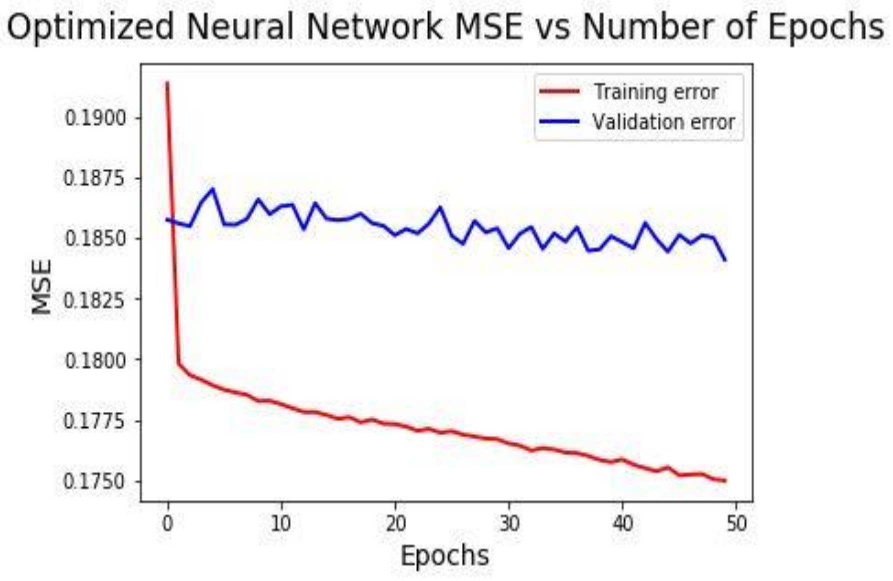

Figure 4. 15: MSE as a function of the number of epochs for 10 hidden neurons after optimization and regularization. This figure shows the effect of using weight decay regularization for reducing overfitting. In comparison with figures 4.9 and 4.10, the validation error is observed to be decreasing past 10 training epochs.

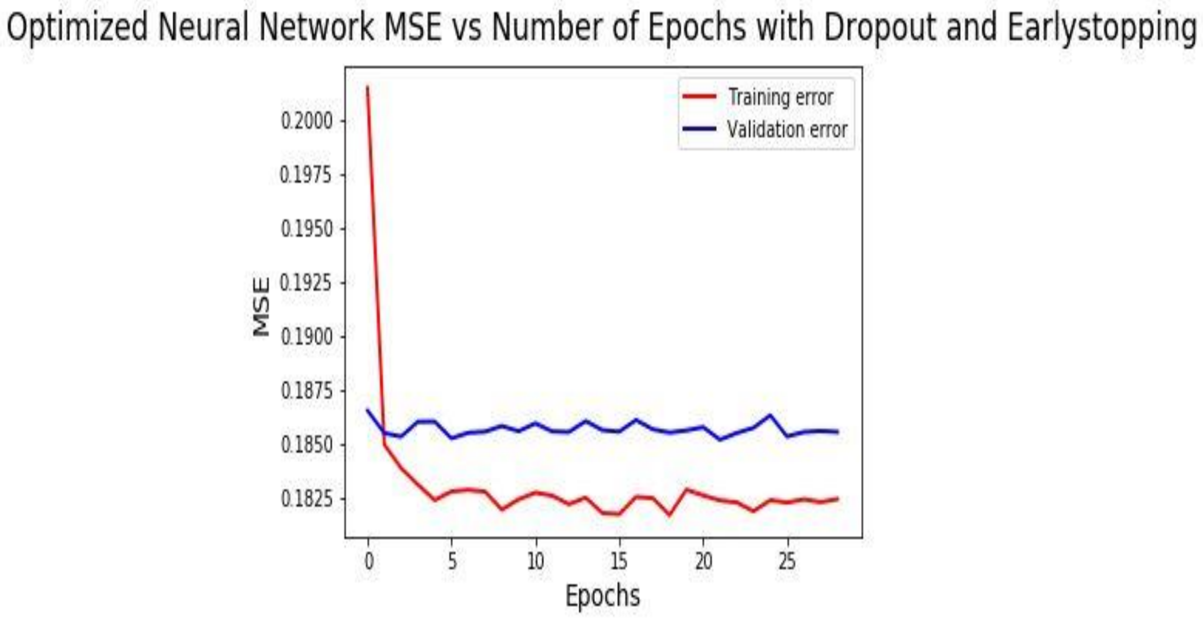

Figure 4. 16: MSE as a function of the number of epochs demonstrating the impact of dropout and early stopping regularization. In contrast to figure 4.15, the training gap between the training error and validation error is reduced due to dropout. Comparing against figures 4.9 and 4.10, the model is no longer demonstrating overfitting.

To further regularize the MLP, *dropout* and *early stopping* were implemented as described in section 3.3.5. Figure 4.16 illustrates the impact of early stopping and dropout. It can be observed in figure 4.16 that there has been a reduction in the error gap between the training and validation error to ~ 1.3% when contrasted with the 6% gap in figure 4.15. This reduction illustrates the regularization impact of dropout, however, some noise has also been introduced into the network; this is also an effect of the dropout regularization according to [71]. The generalisation performance becomes stable after about 5 training epochs, with both errors achieving their minimum value at ~ 27 epochs due to the early stopping.

4.1.5 Linear Support Vector Classification

The Linear Support Vector Classifier (SVC) was also implemented as described in section 3.3.4. using an initial whilst incorporating the chi-squared feature selection (section 3.2.3). The results of the sequential forward search conducted and the generalization performance assessment are presented in figures 4.17 and 4.18 respectively.

Figure 4.17 shows that the optimal number of features which gives the best AUC score is a subset which contains ≥ 32 features, this is because the AUC reaches its maximum value at 32 features and remains at this value until all 62 features have been evaluated. A feature subset containing 34 features was chosen arbitrarily.

Figure 4.18 depicts the learning curve obtained from training the linear SVC model with C=1. The observation from this figure is similar to that observed for the logistic regression model (section 4.1.3), i.e. the model has poor generalization performance for very small training set sizes but as the training set size increases, the training and validation converge, signifying an improving generalization performance. This model also demonstrates high bias as a point is reached (~21000 samples).

A grid search (see section 3.4.2) was conducted to further optimize the Linear SVC model using the regularization parameters: C, l2 penalty term (see 3.3.4) and the 34 selected features. The range of the search was between C=1 and C=120. Only one grid search was required for this model as the optimal parameter was determined from the initial grid-search. The contour plot generated is presented in figure 4.19. As discussed previously, the parameters were optimized with the AUC score which is displayed on the right of the plot, where the red colour signifies the desired parameter region.

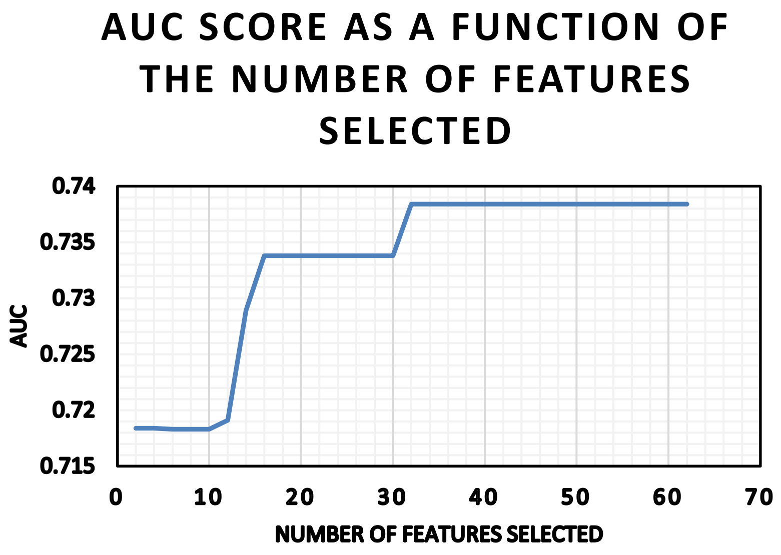

Figure 4. 17: AUC as a function of the number of features selected for the linear SVC model showing the best AUC score is obtained with a feature subset containing ≥ the 32 top features in tables 4.1 and 4.2

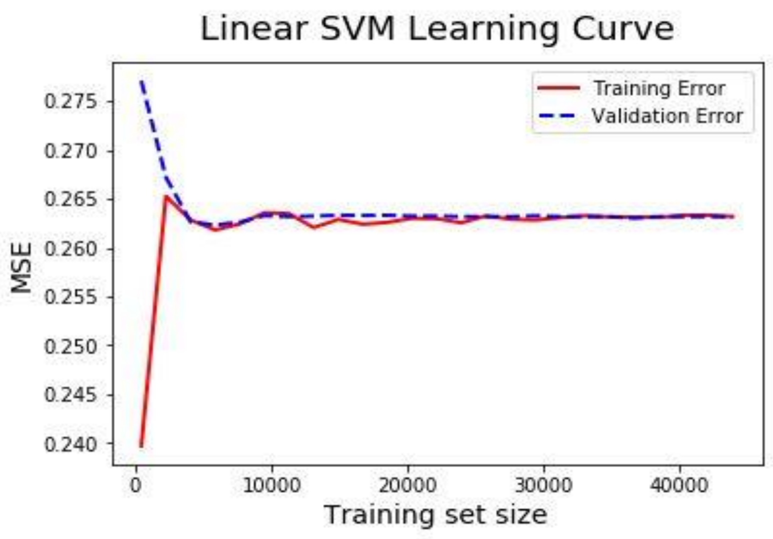

Figure 4. 18: MSE as a function of the training set size for the Linear SVC model. This figure shows for very small training set sizes, the LR model has poor generalization performance and is overfitting the training data. As the training set size increases the generalization performance increases, but both the training and validation error plateau. The model is no longer learning from additional data and demonstrates under fitting for larger training set sizes.

Figure 4.19 shows the best AUC is attained when **C=70**. The SVC model was re-trained with this optimal value and the updated learning curve is presented in figure 4.20. Comparing figures 4.18 and 4.20, there was no significant improvement in the generalization performance or convergence of MSE for the optimized SVC. This implies the SVC model was already very close to its optimal performance on this dataset.

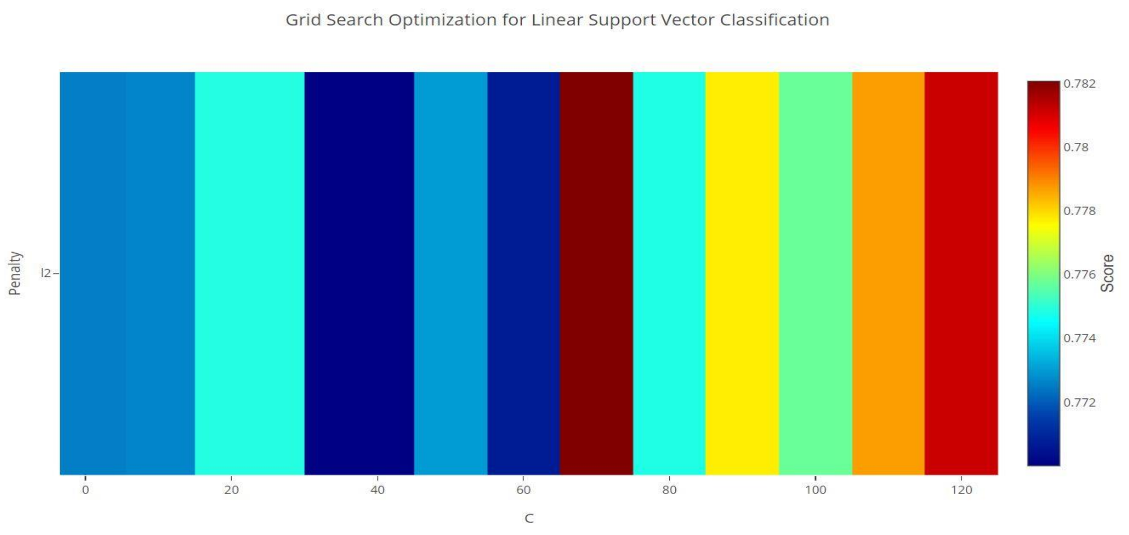

Figure 4. 19: Grid-search contour plot for number of hidden neurons against lambda after chi-square feature selection where the "Score" is the negative MSE being optimized. On the colour bar, red depicts a desired score.

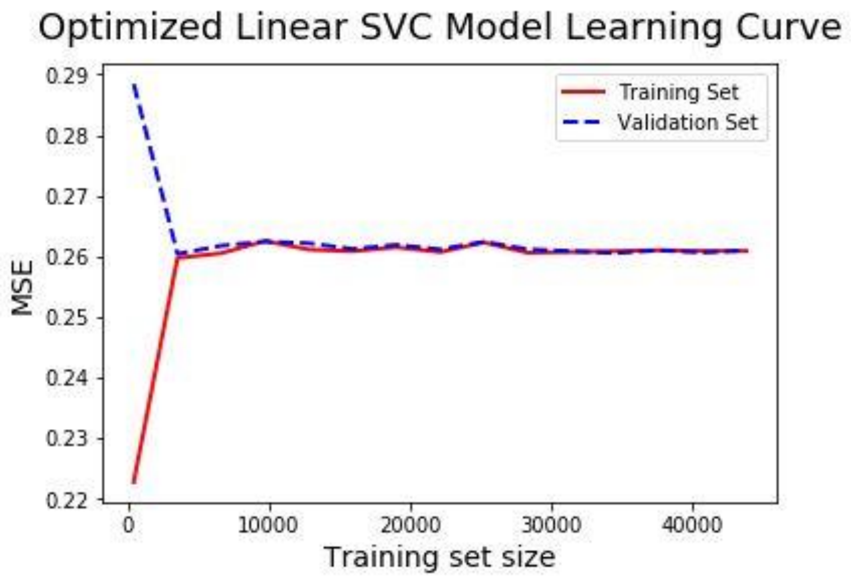

Figure 4. 20: MSE as a function of the training set size for the optimized Linear SVC model. This figure follows a similar trend to figure 4.18, however there is no significant improvement in performance when compared with the initial SVC model in figure 4.18. This implies the SVC model was very close to its optimal performance on this data

4.1.6 Decision Trees

The implementation of the Decision Tree (DT) was as per section 3.3.3. A minimum samples leaf =1 and an unlimited maximum depth were utilised for the initial training of the DT model. An SFS (section 3.2.3) was also implemented and integrated with this initial DT. The result of the SFS is shown in figure 4.21 and 4.22, whilst figure 4.23 illustrates the learning curve produced. From figure 4.22, the optimal feature subset which attains the best AUC score for the DT model is observed to occur when the number of features selected is $\geq$ the top 54 features from tables 4.1 and 4.2 in section 4.1.1. 56 was selected arbitrarily.

The learning curve in figure 4.23 shows the training MSE of the decision tree model is approximately 0 over the entire training set range, although it increases at a small rate as the training set sizes increases. The model is therefore overfitting the training data for very small training set sizes (~1) and for large training almost fits all instances perfectly. However, looking at the validation error for the DT model, for very small training set sizes, the validation error is very high relative to the training error. This implies the model has very poor generalization performance for small training set sizes.

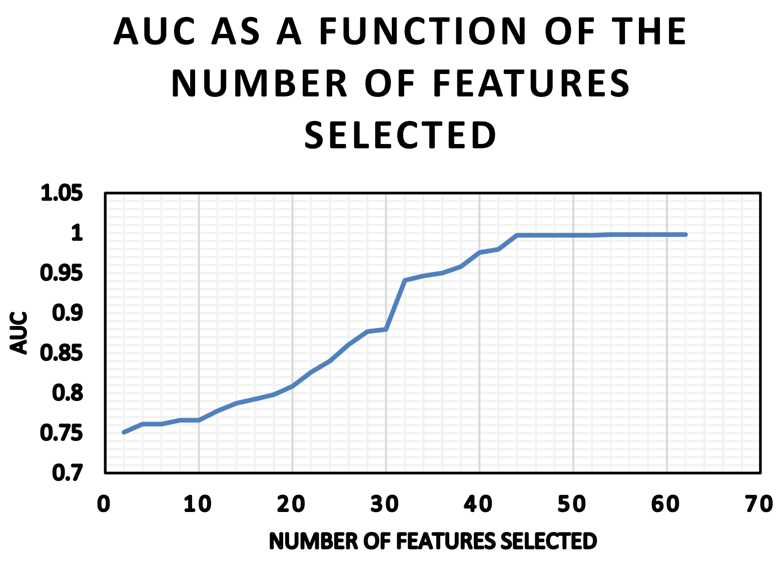

Figure 4. 21: AUC as a function of the number of features selected for the Decision Tree model showing the trend of the SFS (section 3.2.3) conducted.

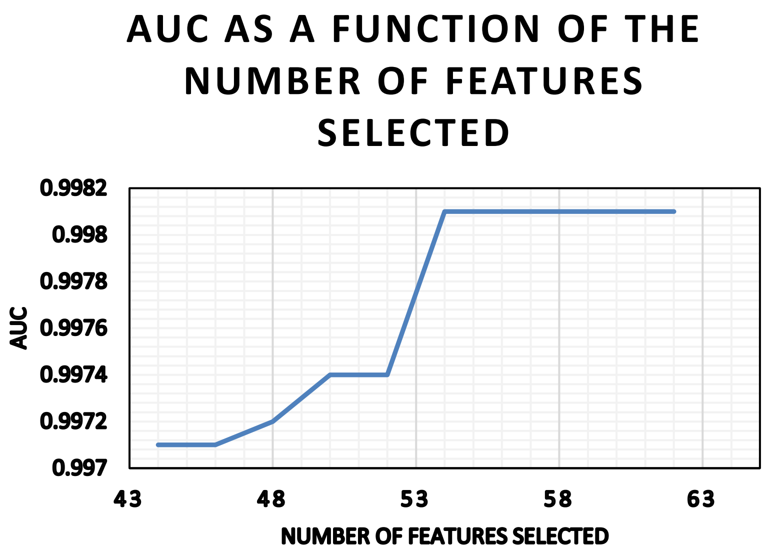

Figure 4. 22: Figure 4.21 zoomed in to show the best AUC score is obtained when a feature subset containing $\geq$ the top 54 features in tables 4.1 and 4.2.

A considerable increase in the generalization performance is observed as the training set size increases due to the validation error converging to a relatively lower value of ~ 0.07. If additional training data was available for training the DT model, we can infer from the trajectories of the training and validation errors that both errors will eventually converge; with the training error ending up at a higher MSE than the MSE shown at ~40000 samples and the validation error ending up lower than the current MSE shown at ~40000 samples.

Due to the overfitting shown by the initial model in figure 4.23, regularization was applied to constrain the model and improve its generalization performance. A wide range grid search with a parameter range of $0\leq maximum\ depth \leq 200$ and $0\leq$ minimum samples leaf$\leq$ 160 was conducted. This was followed by a fine grid-search to select the optimal regularization parameter pairs. Figures 4.24 and 4.25 show the contour plots obtained for the grid-search. From figure 4.24, the grid-search range was narrowed down to the range used in figure 4.25 (a max depth of 40 and min samples leaf of 2 was outputted by the ***"GridSearchCV"*** class for the initial grid search). Figure 4.25 shows the best AUC score is obtained when a maximum depth of 38 and minimum samples leaf of 2 was utilised.

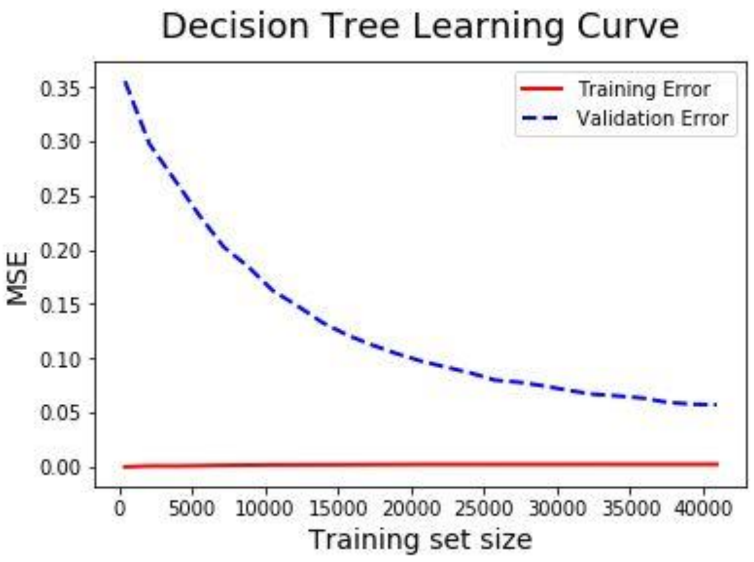

Figure 4. 23: MSE as a function of the training set size for the DT model. This figure shows for small training set sizes, the DT model has poor generalization performance and is overfitting the training data. As the training set increases the generalization performance increases. The trajectories of both the training and validation error imply a convergence will eventually be attained for >40000 training set. However, the training MSE will end up ≥ its value at ~40000 training samples.

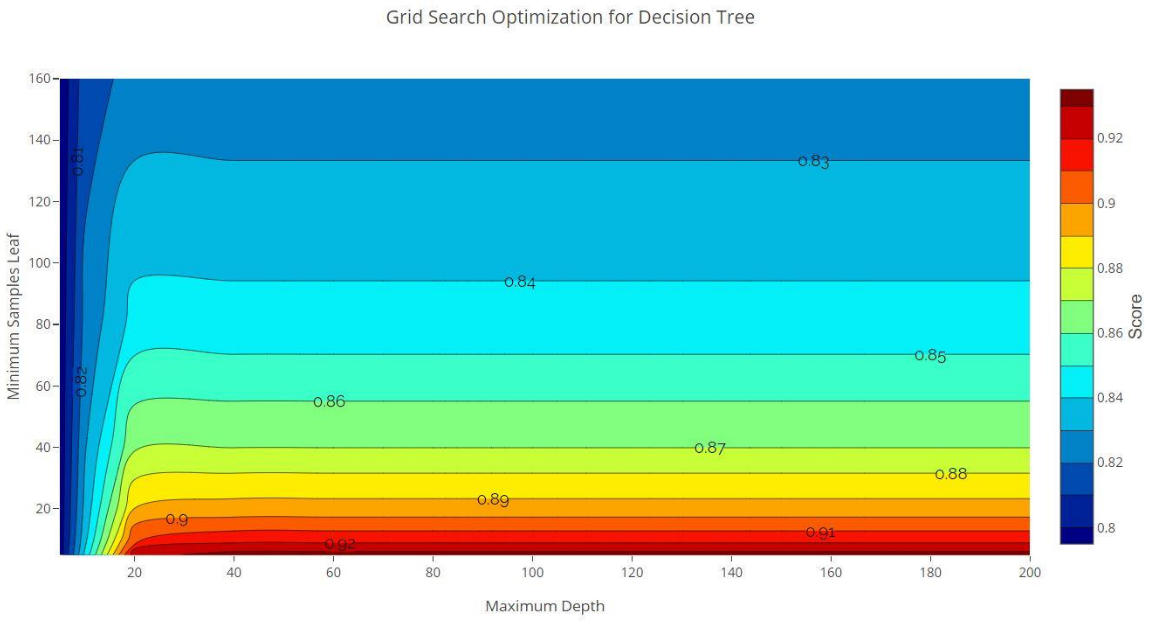

Figure 4. 24: Wide range grid-search contour plot for Minimum samples leaf against maximum depth where the "Score" is the AUC being optimized. On the colour bar, red depicts a desired score.

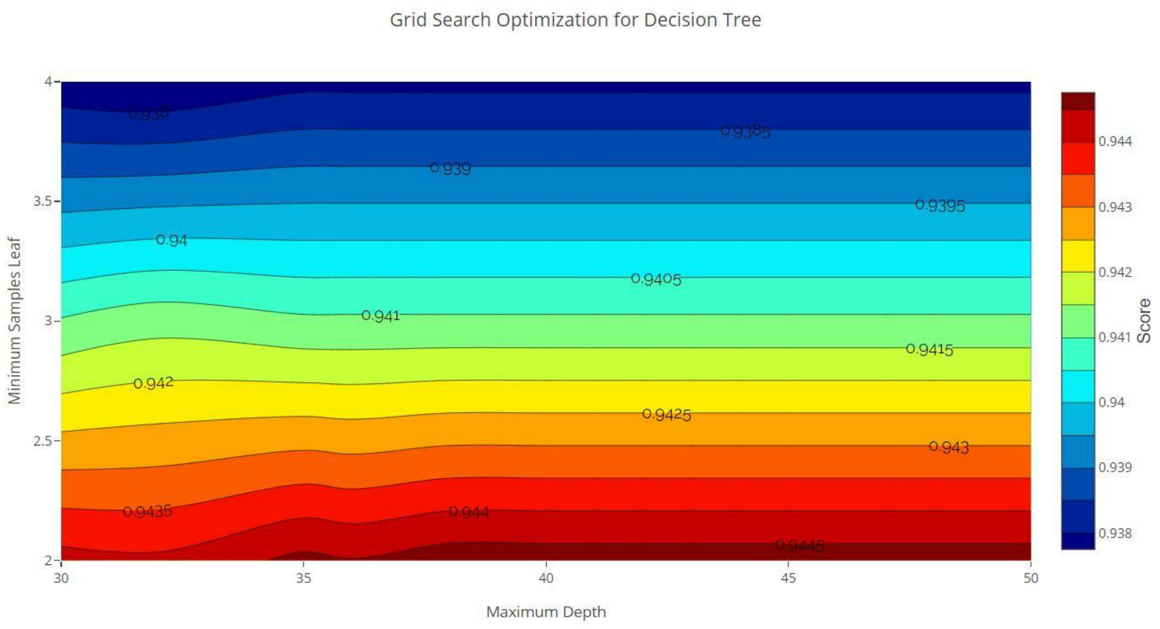

Figure 4. 25: Fine range grid-search contour plot for Minimum samples leaf against maximum depth where the "Score" is the AUC being optimized. On the colour bar, red depicts a desired score.

The learning curve obtained from training the DT model with the regularization parameters obtained from the grid-search is shown in figure 4.26. In this figure, it is observed that the DT model no longer fits very small training set samples perfectly, however as the training set size increases, the training MSE reduces. Therefore, suggesting the DT model is learning from the training data. Comparing the regularized model (figure 4.26) to the un-regularized model in figure 4.23; for small training set sizes, the regularized model has an improved generalization performance because the difference between the training and validation MSE is reduced. The generalization performance also improves with an increase in the training set size. Comparing the trajectories of the regularized model errors against the un-regularized model errors for > 40000 samples, the trajectories of both errors for the regularized model show eventual convergence, however, in this case, the convergence suggest the eventual MSE will be lower when compared with the un-regularized case.

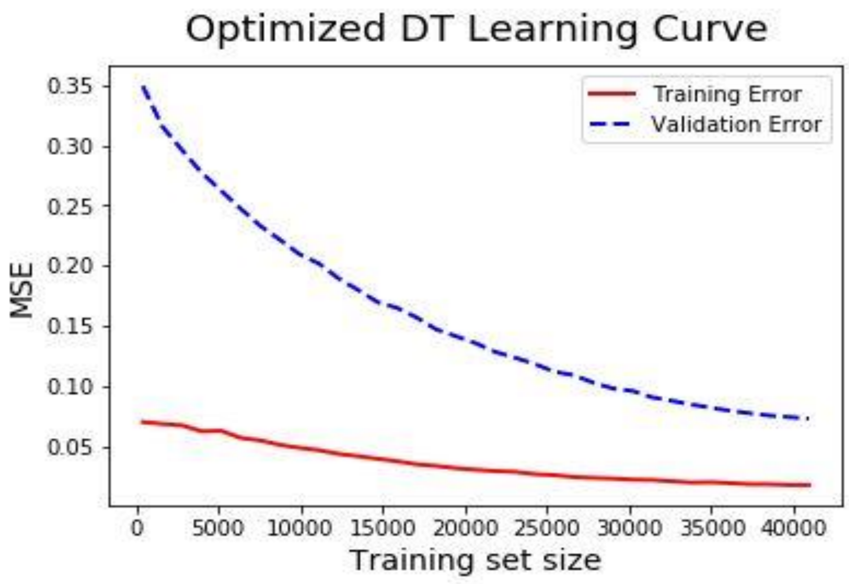

Figure 4. 26: MSE as a function of the training set size for the regularized DT model. This figure shows for small training set sizes; the DT model has an improved generalization performance when compared with the model in figure 4.23.

4.1.7 AdaBoost

An initial AdaBoost algorithm with 50 classifiers in the ensemble was implemented with a Decision Tree of maximum depth =1 as the base classifier (section 3.3.6). An SFS was also carried out using this initial algorithm design. The result of the SFS is displayed in figure 4.27. On inspecting this figure, the best number of features for the AdaBoost algorithm (max depth =1 and number of classifiers = 50) was determined to be 46.

After the use of the SFS for feature selection, the optimal number of classifiers utilised in the ensemble sequence and the learning rate were determined using a 5- fold cross validation wide-range grid-search which was optimized against the AUC score. Figure 4.28 presents the contour plot for the grid-search. The red region as stated before signifies the desired region. On inspecting figure 4.28, it is clear the best number of classifiers to utilise in the AdaBoost algorithm is the maximum of 400 and a learning rate of 1. A brief analysis of the contour plot also shows that the AUC score is likely to be optimised further if the number of classifiers is increased due to upwards diagonal direction of the red colour. However, due to the computational time required for training further classifiers, this maximum number of estimators was selected as the final number for the Adaboost algorithm.

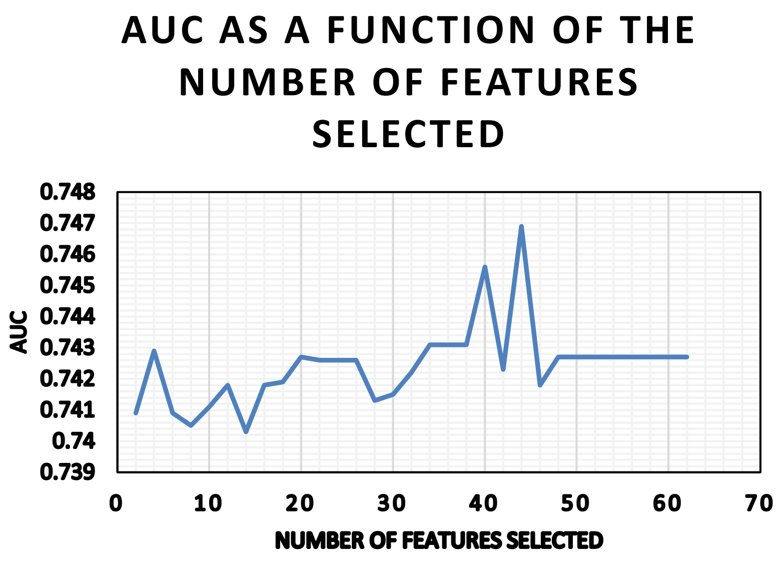

Figure 4. 27: AUC as a function of the number of features selected for the AdaBoost model showing the best AUC score is obtained with a feature subset of the top 46 features in tables 4.1 and 4.2

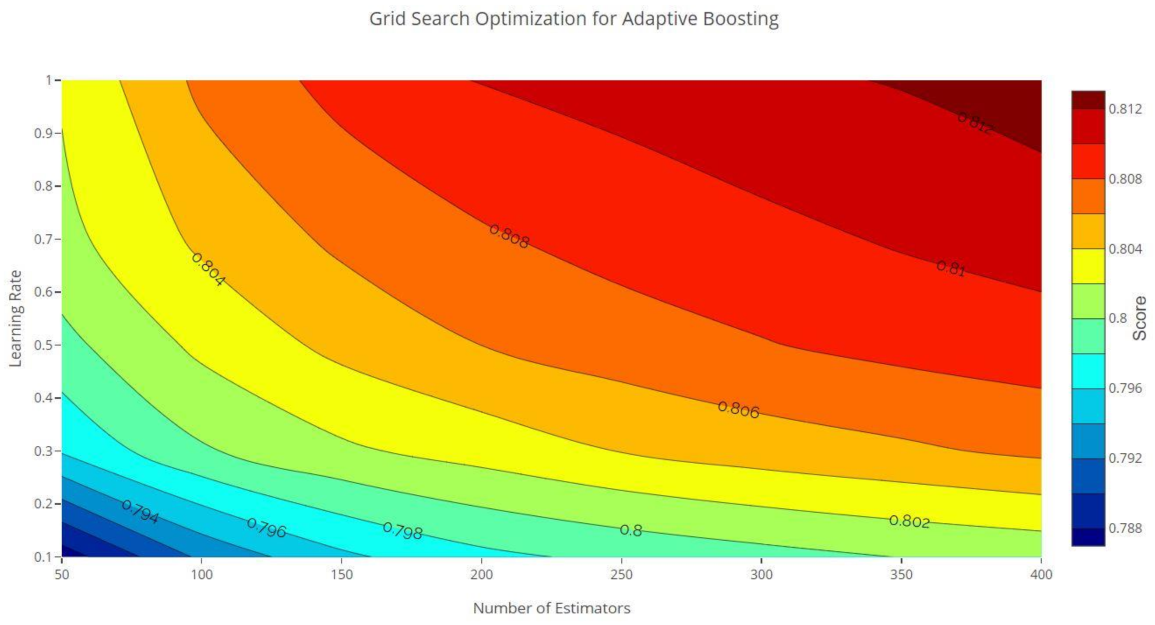

Figure 4. 28: Grid-search contour plot for learning rate against the number of classifiers in the ensemble where the "Score" is the AUC being optimized. On the colour bar, red depicts a desired score.

4.1.8 Test Set Performance Comparison and Voting Classifier

All the optimised algorithms discussed in sub-sections 4.1.2 to 4.1.7 were evaluated on the test set to obtain their final performance. These algorithms excluding the MLP-ANN were combined using *hard voting* as discussed in section 3.3.7 to derive the voting classifier, which was also evaluated on the test set.

Confusion matrices (section 2.7) for all the algorithms are presented in table 4.3, whilst the AUC score which was the main performance measure for comparison (section 3.4.3) was calculated for each algorithm to assess which had the best AUC. The *Sensitivity (Recall)* measure was also calculated for each algorithm to evaluate the 'subscribe' class in isolation as this was the main prediction goal for this work. The recall measure determines how many clients will be accurately predicted to subscribe amongst the true number of clients who actually subscribed. The results of these are presented in table 4.4.

Table 4.3: Test Set Confusion Matrix for the 7 Classifiers with their respective parameters and number of features selected. The values displayed are the averages over 10 runs.

True Class	**Predicted Class**		**Classifier Utilised**	**No. of features**	**Parameters**
	Not subscribed	*Subscribed*			
Not Subscribed	9549 ± 96	1416 ± 96	Gaussian Naïve Bayes	8	N/A
Subscribed	543 ± 22	849 ± 22			
Not Subscribed	9342 ± 70	1623 ± 70	Logistic Regression	46	C= 0.5 Penalty = L1
Subscribed	514 ± 22	878 ± 22			
Not Subscribed	9709 ± 35	1256 ± 35	Decision Tree	56	Max Depth =38 Min Samples Leaf =2
Subscribed	871 ± 15	521 ± 15			
Not Subscribed	9470 ± 28	1496 ± 28	Linear SVC	34	C=70 Penalty = L2
Subscribed	539 ± 18	852 ± 18			
Not Subscribed	9231 ± 125	1734 ± 125	Multilayer Perceptron	52	10 Hidden Units Lambda =0.003
Subscribed	508 ± 23	884 ± 23			
Not Subscribed	9344 ± 70	1621 ± 57	AdaBoost	46	Number of classifiers = 400 Learning rate = 1
Subscribed	500 ± 21	892 ± 21			
Not Subscribed	9446 ± 70	1519 ± 70	Voting Classifier	62	Hard (No weights) Voting
Subscribed	514 ± 22	878 ± 22			

Table 4.4: AUC and Recall Performance measures obtained from the Test set. The values displayed are the Averages over 10 runs.

Performance Measure	GNB	LR	DT	Linear SVC	MLP	AdaBoost	Voting Classifier
Recall	60.99%	63.10%	37.42%	61.23%	63.49%	64.10%	62.98%
AUC	0.7403	0.7415	0.6298	0.738	0.7368	**0.7476**	0.7456

AdaBoost with Duration Attribute

Since the AdaBoost algorithm obtained the best *AUC* and *Recall* score as shown in table 4.4, it was re-implemented *with the "Duration"* attribute for the purpose of literature comparison only. The same parameters and methods for the initial AdaBoost classifier *without the "Duration"* attribute in section 4.1.7 were implemented. In this case, the Sequential Forward Search obtained the best AUC score with the top ***24*** features from tables 4.1. After evaluating on the Test set for 10 runs, an average *AUC score* of **0.8825** was obtained and a *Recall* of **0.8917**.

Figure 4.29 presents a bar graph comparison of the AUC score for all the 7 classifiers evaluated and the AdaBoost classifier *with the Duration attribute*.

4.2 Discussion

Looking at Table 4.3 and 4.4, the best *Recall* score was attained by the AdaBoost, correctly predicting 64.1% of the total number of clients who actually subscribed whilst using ~74% of the maximum number of features. The MLP, LR and Voting classifier followed closely in that order, attaining a recall of 63.49%, 63.10% and 62.98% respectively. The GNB model used the lowest percentage of features at ~13%, followed by the SVC at ~55%. Both of these produced similar predictions for the 'subscribed' class. The Decision Tree obtained the lowest recall score.

Comparing only the initial 5 parametric and non-parametric classifiers from figure 4.29, the AUC scores obtained for all 5 classifiers were close. The LR and GNB models attained the highest scores of 0.7415 and 0.7403 respectively, followed by the Linear SVC, MLP and then the DT.

Now considering the whole of figure 4.29 i.e. including the ensemble classifiers, it is observed that the AdaBoost classifier achieved the best AUC score amongst the 7 classifiers evaluated with an AUC of 0.7476. Therefore, on the basis of overall classification performance, this is the best classifier. This score was closely followed by the Voting classifier which achieved 0.7456.

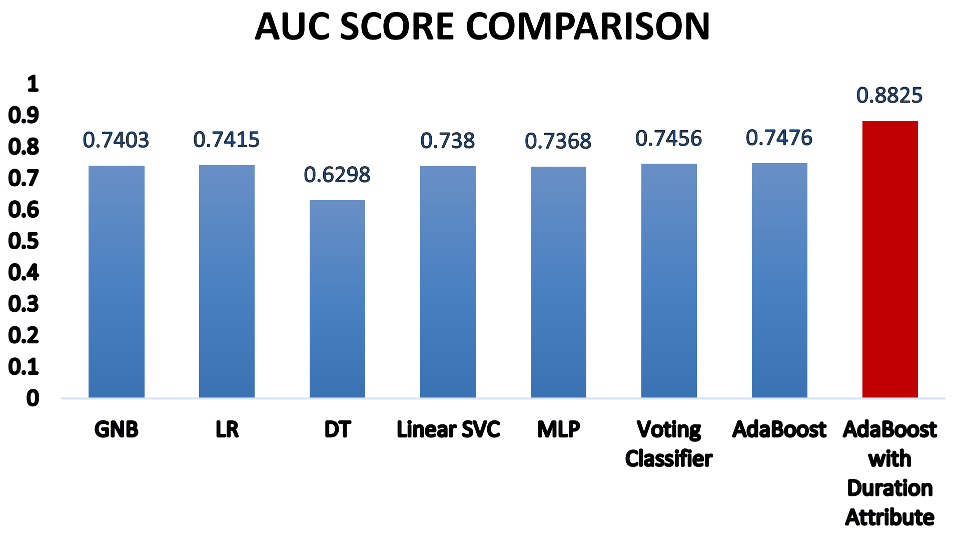

Figure 4. 29: AUC score comparison for all evaluated classifiers. The AdaBoost with Duration attribute is NOT part of this work and is only include for comparison only; it is coloured in red to denote this.

Overall, the aims and objectives of this work stipulated section 1.1 were achieved. The performance of parametric and non-parametric methods which incorporated feature selection using the chi-square test were compared. The results show that with the methodologies employed, parametric methods perform slightly better than non-parametric methods for the bank marketing classification problem. Considering the learning curves for each classifier, the non-parametric methods seem to have relatively higher gaps between their training and validation error even after regularization. Therefore, the obtained results could be as a consequence of the non-parametric methods having too many degrees of freedom even after regularization and therefore not generalizing well on unseen instances relative to the parametric methods.

It is also clear that the ensemble methods obtained the superior performance in comparison to the parametric and non-parametric methods. The AdaBoost boosted the performance of the Decision Tree which was the weakest classifier whilst the Voting classifier leveraged the performance of the parametric and non-parametric methods to increase performance.

In addition to meeting the stipulated objectives, when the results obtained are compared with the literature surveyed in table 2.1, the AUC score of the AdaBoost places 4th in the list with an approximate difference of 20% from 1st place. However, it should be recalled that this work was carried out **without** the duration attribute. If the Duration attribute was utilised, the AdaBoost classifier would have obtained an AUC score of 0.8825 as shown in figure 4.29, an 18% improvement on the score obtained without the attribute. This improved score would place the AdaBoost classifier 3rd in the list but with only a 5% difference from 1st place. These results confirm the impact of the duration attribute on the bank marketing classification.

5 CONCLUSIONS

The Direct marketing technique employed by banks to promote their products and services can prove very effective to elicit subscriptions from clients if managed properly. This work evaluated different parametric, non-parametric and ensemble methods for performing classification on the bank marketing dataset whilst incorporating a filter feature selection strategy implemented using a sequential forward search.

In regards to pre-processing, the bank marketing dataset was shown to be imbalanced, with the ’not subscribed’ class being the majority class. This imbalance would have affected the performance of models such as Decision trees or the MLP, so up sampling was performed to balance the dataset. In addition, because the dataset was imbalanced, the accuracy performance measure was deemed to be ineffective for comparing performance of the classifiers. Furthermore, the dataset was shown to be ~89% linearly separable which meant linear models could also be evaluated. Feature selection obtained at least a 10% reduction in the number of features required for training in most classifier cases and in the GNB case, a 92% reduction was obtained.

On the initial training of the classifiers, overfitting and poor generalization performance was exhibited for small training set sizes, this was followed by an increase in generalization performance as the training set sizes increased. The most overfitting was exhibited by the non-parametric classifiers, especially the MLP. Grid-search cross validation was utilised for selecting the regularization parameters of the classifiers which helped to improve generalization performance for some of the classifiers, where as in some cases, there was no significant improvement.

Following the evaluation of the optimized models on unseen data instances, the obtained results showed that the parametric methods performed slightly better than the non-parametric methods, although with relatively similar AUC results. This result was attributed to better generalization performance of the parametric methods as seen from the optimized learning curves with the non-parametric methods tending to over fit the training data even after regularization.

The results also showed that ensemble methods indeed improve the results of individual classifiers. AdaBoosting significantly improved the AUC performance of the Decision tree non-parametric classifier by approximately 19%, whilst the voting ensemble method combined both parametric and non-parametric methods to obtain an improved performance of 1% from the best parametric method. Overall the AdaBoost method obtained the best AUC classification score of 0.7476.

This work evaluated realistic ensemble models against the AUC metric with the exclusion of the duration attribute. It therefore brings novelty to the bank marketing research space as similar research on the bank marketing dataset had utilised the posterior duration attribute in their models and also utilised accuracy as the performance metric, both of which being ineffective methods for proper evaluation. Overall, this work was comparable to the literature regarding the bank marketing dataset. The AdaBoost was implemented with the duration attribute for *comparison purposes only*, attaining an AUC of 0.8625. Therefore, placing 3rd (5% from 1st) amongst the surveyed literature which utilised the duration attribute.

Even though the AdaBoost algorithm obtained the best AUC score, the grid-search results showed that it could have been further optimised by utilising more classifiers in the ensemble sequence. Due to time and computational constraints, this optimization was not attempted, but can be implemented in the future when there is more flexibility on time and computational resources.

The Neural Network was also not implemented as part of the voting classifier due to compatibility issues between the Keras [74] and Scikit-learn [66] implementation libraries. Furthermore, weighted voting could also have been implemented as this performs better than the no weights method [39] implemented in this research.

Other boosting techniques such as Stacking [76] could be used instead of the voting or AdaBoosting techniques for optimising the performance of individual classifiers.

6 REFERENCES

[1] C. Merz, P. Murphy, D. Aha," Bank marketing Data Set" in *UCI Machine Learning Repository,* June, 2014. [Online]. Available : https://archive.ics.uci.edu/ml/datasets/Bank+Marketing. Accessed on: Dec, 1, 2017.

[2] G.J. Holy and S.J. Mann, "The Adoption of Marketing by Financial Institutions in the UK", *The Service Industries Journal*, vol. 8, no. 4, p. 497, Jul. 2006. [Online]. DOI: 10.1080/02642068800000067

[3] P. W. Turnbull and M. L. Gibbs, "Marketing Bank Services to Corporate Customers: The Importance of Relationships", *International Journal of Bank Marketing,* vol. 5, no. 1, pp.19-26, 1987. [Online]. DOI: 10.1108/eb010796

[4] C. L. Bauer and J. Miglautsch, "A conceptual definition of Direct Marketing", *Journal of Direct Marketing*, vol. 6, no. 2, pp. 8-13, Spring. 1992. [Online]. DOI: 10.1002/dir.4000060204

[5] P. Wang and L. A. Petrison, "Direct Marketing activities and personal privacy: A consumer survey", *Journal of Direct marketing,* vol .7, no. 1, pp. 1-13, Winter. 1993. [Online]. DOI: 10.1002/dir.4000070104

[6] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997

[7] H. Li, D. Parikh, Q. He, B. Qian, Z. Li, D. Fang and A. Hampapur, "Improving rail network velocity: A machine learning approach to predictive maintenance", *Transportation Research Part C*. vol. 45, no. 1, pp. 17-26. Aug. 2014. [Online]. DOI: 10.1016/j.trc.2014.04.013

[8] I. Yeh, C. Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients", *Expert Systems with Applications*, vol. 36, no. 1, pp. 2473–2480, 2009. [Online]. DOI: 10.1016/j.eswa.2007.12.020

[9] E. Alpaydin, *Introduction to Machine Learning,* 2nd ed. Cambridge, Massachusetts: The MIT Press, 2010

[10] X. Zhu and A. B. Goldberg, *Introduction to Semi-Supervised Learning.* San Rafael, California: Morgan & Claypool, 2009

[11] S. Moro, P. Cortez and R. M. S. Laureano, “Enhancing bank direct marketing through data mining”, in *proceedings of the 41st International Conference of the European Marketing Academy, 2012,* pp. 1-8, May. 2012. [Online]. Available: https://www.researchgate.net/publication/236231155_Enhancing_Bank_Direct_Marketing_through_Data_Mining

[12] S. Moro, P. Cortez and P. Rita, “A Data-Driven Approach to Predict the Success of Bank Telemarketing”, *Decision Support Systems,* vol. 62, no. 1, pp. 22-13, Jun. 2014. [Online]. DOI: 10.1016/j.dss.2014.03.001

[13] Curtis L. Lentz, “Data Analytics Approaches to Predicting the Success of Bank Telemarketing”, Master Thesis, Central Connecticut State University, 2015.

[14] K. Kim, C. Lee, S. Jo and S. Cho, “Predicting the success of Bank Telemarketing using Deep Convolutional Neural Network”, in *proceedings of the 7th International Conference of Soft Computing and Pattern Recognition 2015,* Fukuoka, Japan, pp. 1-4. [Online]. DOI: 10.1109/SOCPAR.2015.7492828

[15] S. Lahmiri, “A two-step system for direct bank telemarketing outcome classification.”, *Intelligent Systems in Accounting, Finance and management,* vol. 24, pp. 40-55, 2017. [Online]. DOI: 10.1002/isaf.1403

[16] S. B. Kotsiantis, D. Kanellopoulos and P. E. Pintelas, “Data Pre-processing for Supervised Learning”, International Journal of Computer Science, vol. 1, no. 2, pp. 111-117, Nov. 2006. [Online]. Available: https://www.researchgate.net/

[17] C. Carrie, “Using decision trees to improve case-based learning”, in *Proceedings of the 10th International Conference on Machine Learning 1993,* University of Massachusetts, Amherst, pp. 25-32. [Online]. Available : http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.30.1899&rep=rep1&type=pdf

[18] M. Singh and G. M. Provan. “Efficient learning of selective Bayesian network classifiers. In Machine Learning”, in *proceedings of the 13th International Conference on Machine Learning 1996,* Bari, Italy. pp. 453-461. [Online]. Available: https://dl.acm.org/

[19] R. Setiono and H. Liu, “Neural-network feature selector”, *IEEE Trans. Neural Networks,* vol. 8, no. 3, pp. 654–662, May. 1997. [Online]. DOI: 10.1109/72.572104

[20] D. Elizondo, “The Linear Separability Problem: Some Testing Methods”, *IEEE Transaction on neural networks,* vol. 17, no. 2, p. 330-344, Mar. 2005. [Online]. DOI: 10.1109/TNN.2005.860871

[21] V. N. Vapnik, *The nature of Statistical Learning Theory.* 2nd ed, Berlin, Germany: Springer Varlag, 2000.

[22] S. German, E. Bienenstock and R. Doursat, "Neural networks and the bias/variance dilemma", *Neural Computation*, vol. 4, no. 1, pp. 1–58, Jan. 1992. [Online]. DOI: 10.1162/neco.1992.4.1.1

[23] A. Keles, A. Keles, “IBMMS Decision Support Tool for management of Bank Telemarketing Campaigns”, *International Journal of Database Management Systems,* vol. 7, no. 5, pp. 1-15, Oct. 2015. [Online]. DOI: 10.5121/ijdms.2015.7501

[24] C. Yang, “Predicting Success of Bank Telemarketing with Classification Trees and Logistic Regression”. Master Thesis, University of Texas at, Austin, Texas, 2016.

[25] H. Zhang, J, Su, “Naïve Bayesian Classifiers for Ranking*”,* in *proceedings of the 15th European Conference on Machine Learning 2004,* Pisa, Italy. pp. 501-512. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-30115-8_46

[26] C. J Peng, K. L. Lee and G. M. Ingersoll, “An Introduction to Logistic Regression Analysis and Reporting”, *The Journal of Educational Research,* vol. 96, no. 1, pp. 3-14, Apr. 2002. [Online]. DOI: 10.1080/00220670209598786

[27] H. Nielesen, “Theory of Back Propagation Neural Network” in *International 1989 Joint Conference on Neural Networks*, Washington, DC, USA. pp. 593-605. [Online]. Available: https://ieeexplore.ieee.org/document/118638/

[28] Y. Kumar and G. Sahoo, “Analysis of Parametric & Non Parametric Classifiers for Classification Technique using WEKA”, *International Journal of Information Technology and Computer Science,* vol. 7, no. 1, pp. 43-49, Jul. 2012. [Online]. DOI: 10.5815/ijitcs.2012.07.06

[29] X. Geng, D.Zhan and Z.Zhou, “Supervised Nonlinear Dimensionality Reduction for Visualization and Classification”, *IEEE Transactions on Systems, Man and Cybernetics-*

Part B: Cybernetics, vol.35, no. 6, pp. 1098-1099, Dec. 2005. [Online]. DOI: 10.1109/TSMCB.2005.850151

[30] G. Cauwenberghs and T. Poggin, “Incremental and Decremental Support Vector Machine Learning” in *proceedings of the 13th International Conference on Neural Information Processing Systems 2000,* Denver, CO, USA. pp. 388-394 [Online]. Available: https://dl.acm.org/citation.cfm?id=3008751.3008808

[31] Y. Saeys, I. Inza and P. Larranaga, “A review of feature selection techniques in bioinformatics”, *Bioinformatics,* vol.23, no. 19, pp. 2507-2517, Oct. 2007. [Online]. DOI: 10.1093/bioinformatics/btm344

[32] M. A. Friedl and C. E. Brodley, “Decision Tree Classification on Land Cover from Remotely Sensed Data”, *Remote Sensing Environment,* vol. 61, no.3, pp. 399-404, Sept. 1993. [Online]. DOI: 10.1016/S0034-4257(97)00049-7

[33] L. K. Hansen and P. Salamon, “Neural Network Ensembles”, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 12, no. 10, pp. 993-1001, Oct. 1990. [Online]. DOI: 10.1109/34.58871

[34] T. Chai and R. R. Draxler, “Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature”, *Geoscientific Model Development,* vol.7, no.1, 7, pp. 1247–1250, Jun. 2014. [Online]. DOI: 10.5194/gmd-7-1247-2014

[35] A. Jain and D. Zongker, “Feature Selection: Evaluation, Application, and Small Sample Performance”, *IEEE Transactions Pattern Analysis and Machine Intelligence,* vol.19, no.2, pp. 153-158, Feb. 1997. [Online]. DOI: 10.1109/34.574797

[36] A. Janecek, W. Gansterer, M. Demel and G. Ecker, “On the Relationship Between Feature Selection and Classification Accuracy”, in *proceedings of the Workshop on New Challenges for Feature Selection in data Mining and Knowledge Discovery at ECLM/PKDD 2008,* Antwerp, Belgium. Pp. 90-105 [Online]. Available: http://proceedings.mlr.press/v4/janecek08a.html

[37] A. Widodo and B. Yang, “Application of nonlinear feature extraction and support vector machines for fault diagnosis of induction motors”, *Expert Systems with Applications,* vol. 33, no.1, pp. 241-250, Jul. 2007. [Online]. DOI: 10.1016/j.eswa.2006.04.020

[38] S. Fortmann-roe. “Understanding the Bias-Variance Tradeoff”, in *scott.fortmann-roe,* Jun. 2012. [Online]. Available: http://scott.fortmann-roe.com/docs/BiasVariance.html Accessed on: May 7, 2018.

[39] A. Geron, *Hands-On Machine Learning with Scikit-Learn and* TensorFlow, Sebastopol, CA, USA: O’Reilly, 2017.

[40] C. Campbell and Y. Ting, *Learning with Support Vector Machines*, San Rafael, California: Morgan & Claypool, 2011.

[41] P. Harrington, “Support Vector Machine Classification Trees”, *Analytical Chemistry,* vol. 87. no. 21, pp. 11065-11071, Oct. 2015. [Online]. DOI: 10.1021/acs.analchem.5b03113

[42] Y. Coadou, “Decision Trees”, *EPJ Web of Conferences,* vol. 4, no. 02003, pp. 1-18, Apr. 2010. [Online]. DOI: 10.1051/epjconf/20100402003

[43] J. Asare-Frempong and M. Jayabalan, “Predicting Customer Response to Bank Direct Telemarketing Campaign”, *2017 International Conference on Engineering Technologies and Technopreneurship (ICE2T),* Kuala Lumpur, Malaysia, pp. 1-4. [Online]. DOI: 10.1109/ICE2T.2017.8215961

[44] T. Dietterich, “Ensemble Methods in Machine Learning” in *First International Workshop, MCS 2000.* Cagliari, Italy, vol. 1857, pp. 1-15. DOI: 10.1007/3-540-45014-9_1

[45] T. Parlar and S. Acaravci, “Using Data Mining Techniques for Detecting the Important Features of the Bank Direct Marketing Data”, *International Journal of Economics and Financial Issues,* vol.7, no. 2, pp. 692-696, 2017. [Online]. Available: http://dergipark.gov.tr/download/article-file/365990

[46] L. Sing’oei and J. Wang, “Data mining framework for direct marketing: A case study of bank marketing”, *International Journal of Computer Science Issues*, vol. 10, no. 2, pp. 198-203, Mar. 2013. [Online]. Available: https://pdfs.semanticscholar.org/3b09/589fc06dc60c5594da1708da276d0e22287b.pdf

[47] Y. Pan and Z. Tang, “Ensemble methods in bank direct marketing”, in *2014 11th International Conference on Service Systems and Service Management (ICSSSM), Beijing, China, pp. 1-5. [Online]. DOI:* 10.1109/ICSSSM.2014.6874056

[48] O. Apampa, "Evaluation of Classification and Ensemble Algorithms for Bank Customer Marketing Response Prediction", *Journal of International Technology and Information Management,* vol. 25, no. 4, pp. 85-100, 2016. [Online]. Available: http://scholarworks.lib.csusb.edu/jitim/vol25/iss4/6

[49] Y. Freund and R. Schapire, “A decision-theoretic generalization of on-line learning and application to boosting”, *Journal of Computer and System Sciences*, vol. 55, no.1, pp. 119-139, Aug. 1997. [Online]. DOI: 10.1006/jcss.1997.1504

[50] A. Estabrooks, T. Jo and N. Japcowicz, “A multiple resampling method for learning from imbalanced datasets”, *computer intelligence,* vol. 20, no. 1, pp. 18-36, Feb. 2004. [Online]. DOI: 10.1111/j.0824-7935.2004.t01-1-00228.x

[51] H. Mohamed, N. Abdelazim, M. Zahran and O. Saavedra, “Assessment of Artificial Neural Network for bathymetry estimation using High Resolution Satellite imagery in Shallow Lakes: Case Study El Burullus Lake.” in *Eighteenth International Water Technology Conference, IWTC18, Sharm ElSheikh, Egypt,* pp. 1-12. [Online]. Available : https://www.researchgate.net/publication/273768094_Assessment_of_Artificial_Neural_Network_for_bathymetry_estimation_using_High_Resolution_Satellite_imagery_in_Shallow_Lakes_Case_Study_El_Burullus_Lake

[52] B. Jantawan and C. Tsai, “Comparison of Filter and Wrapper Approaches with Data Mining Techniques for Categorical Variables Selection”, *International Journal of Innovative Research in Computer and Communication Engineering,* vol.2, no. 6, pp. 4501-4508, Jun. 2014. [Online]. Available : http://www.rroij.com/open-access/a-comparison-of-filter-and-wrapperapproaches-with-data-mining-techniques-forcategorical-variables-selection.php?aid=46225

[53] G. Dougherty, *Pattern Recognition and Classification*, New York City, USA: Springer-Verlag New York, 2013

[54] *R. Fan, K. Chang, C. Hsieh, X. Wang and C. Lin, "LIBLINEAR: A library for large linear classification", Journal of Machine Learning Research, vol. 9, no.1 , pp. 1871-1874, Aug. 2008. [Online]. Available : https://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf*

[55] S. J. Wright, "Coordinate descent algorithms", *Mathematical Programming*, vol. 151, no. 1, pp. 3-34, Jun. 2015. [Online]. DOI: 10.1007/s10107-015-0892-3

[56] D. G. T. Denison, B. K. Mallick and A. F. M. Smith, "A Bayesian CART algorithm", *Biometrika, vol. 85, no.2, pp. 363-377, Jun. 1998. [Online]. DOI: 10.1093/biomet/85.2.363*

[57] B. Scholkopf, K. Sung, C.J.C. Burges, F. Girosi, P. Niyiogi, T. Poggio and V. Vapnik, "Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers", *IEEE Transactions on Signal Processing,* vol. 45, no. 2, pp. 2758-2765, Nov. 1997. [Online]. DOI: 10.1109/78.650102

[58] L. Wang, J. Zhu and H. Zou, "THE DOUBLY REGULARIZED SUPPORT VECTOR MACHINE", *Statistica Sinica*, vol. 16, No. 2, pp. 589-615, Apr. 2006. [Online]. Available : http://www3.stat.sinica.edu.tw/statistica/oldpdf/A16n214.pdf

[59] S. Rosset, J. Zhu and T. Hastie, "Boosting as a regularized path to a maximum margin classifier", *Journal of Machine Learning Research,* vol. 5, no.1, pp. 941–973, Jan. 2004. [Online]. Available : http://www.jmlr.org/papers/volume5/rosset04a/rosset04a.pdf

[60] D. P. Kingma and J. Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION", in *the 3rd International Conference for Learning Representations,* San Diego, 2015, pp. 1-15. [Online]. Available: https://arxiv.org/abs/1412.6980

[61] K. Gurney, *An Introduction to Neural Networks*, Boca Raton, FL: CRC Press, 1997.

[62] J. A. Hertz and A. Krogh, "A Simple Weight Decay Can Improve Generalization", in *Neural Information Processing Systems 1991,* pp. 950-957. [Online]. Available : https://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization

[63] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout : A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research,* vol. 5, no.1, pp. 1929–1958, Nov. 2013. [Online]. Available : http://jmlr.org/papers/v15/srivastava14a.html

[64] J. Zhu, S. Rosset, H. Zou and T. Hastie. "Multi-class AdaBoost", *Statistics and Its Interface,* vol.2, no. 1, pp. 349-360, 2009. [Online]. Available : https://web.stanford.edu/~hastie/Papers/SII-2-3-A8-Zhu.pdf

[65] F. Provost and T. Fawcett, *Data Science for Business*, Sebastopol, CA, USA: O'Reilly, 2013.

[66] Pedregosa et al., "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, vol. 12, no.1, pp. 2825–2830, Oct. 2011. [Online]. Available: http://www.jmlr.org/papers/v12/pedregosa11a.html

[67] C. Hsu, C. Chang, and C. Lin, "A practical guide to support vector classification," National Taiwan University, Taiwan, Technical Report, Apr. 2010.

[68] H. He and E. Garcia, "Learning from Imbalanced Data", *IEEE Transaction on Knowledge and Discovery,* vol. 21, no. 9, pp. 1263-1281, Sept. 2009. [Online]. Available : https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5128907

[69] H. Guo and H.L. Viktor, "Learning from Imbalanced Data Sets with Boosting and Data Generation: The DataBoost IM Approach," *ACM SIGKDD Explorations Newsletter,* vol. 6, no. 1, pp. 30-39, Jun. 2004. [Online]. DOI: 10.1145/1007730.1007736

[70] M.A. Maloof, "Learning When Data Sets Are Imbalanced and When Costs Are Unequal and Unknown," in Workshop Learning from Imbalanced Data Sets II, ICML, Washington DC, 2003, pp. 1-7. [Online]. Available: https://pdfs.semanticscholar.org/9b8a/d2d62044864b8d1dc0e06bcb1045846c9093.pdf

[71] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors". University of Toronto, Canada, Technical Report, Jul. 2012.

[72] H. Mann and A. Wald, "On the Choice of the Number of Class Intervals in the Application of the Chi Square Test", *The Annals of Mathematical Statistics,* vol.13, no. 3, pp. 306-317, Sep. 1942. [Online]. Available: http://www.jstor.org/stable/2235942

[73] D. Grzonka, G. Suchack and B. Borowik, “Application of Selected Supervised Classification Methods to Bank Marketing”, *Information Systems in Management*, vol. 5, no. 1, pp. 36-48, Jun. 2016. [Online]. Available : https://www.researchgate.net/publication/303857692

[74] F. Chollet et al., “Keras”, 2015. Available : Keras.io

[75] Plotly technologies Inc., *Collaborative data science*, Montréal, QC : Plotly Technologies Inc., 2015 [Online]. Available : https://plot.ly

[76] D. Wolpert, “Stacked Generalization”, *Neural Networks*, vol. 5, no. 2, pp. 241-259. 1992. [Online]. DOI: 10.1016/S0893-6080(05)80023-1

7 PROJECT MANAGEMENT

Overall the project was executed within the stipulated timeframe as the Final Report was handed in on time.

Looking at Semester 1 in figure 7.1, the Aims and objectives finalisation, Investigation of machine learning libraries and writing of the interim report were all concluded on schedule. However, the Literature review and Tutorial phases overran their scheduled duration by 9 and 4 weeks respectively. The broad landscape of machine learning meant that I had to undertake more research than expected, in addition, my initial software knowledge was not up to par with the expertise required for a machine learning project, this led to additional time in undertaking the tutorials for implementation software required for this project.

For Semester 2, the carried over tasks from semester 1 were completed by week 15. Out of the 6 tasks required for completion in Semester 2, 50% were overrun. These were the Data pre-processing, algorithm training and analysis of results. These are very key to the implementation of a machine learning project; therefore, more time was spent on optimizing the outcomes of each of these tasks. The performance evaluation and Final report writing were completed on schedule, whilst the oral viva task is still in progress.

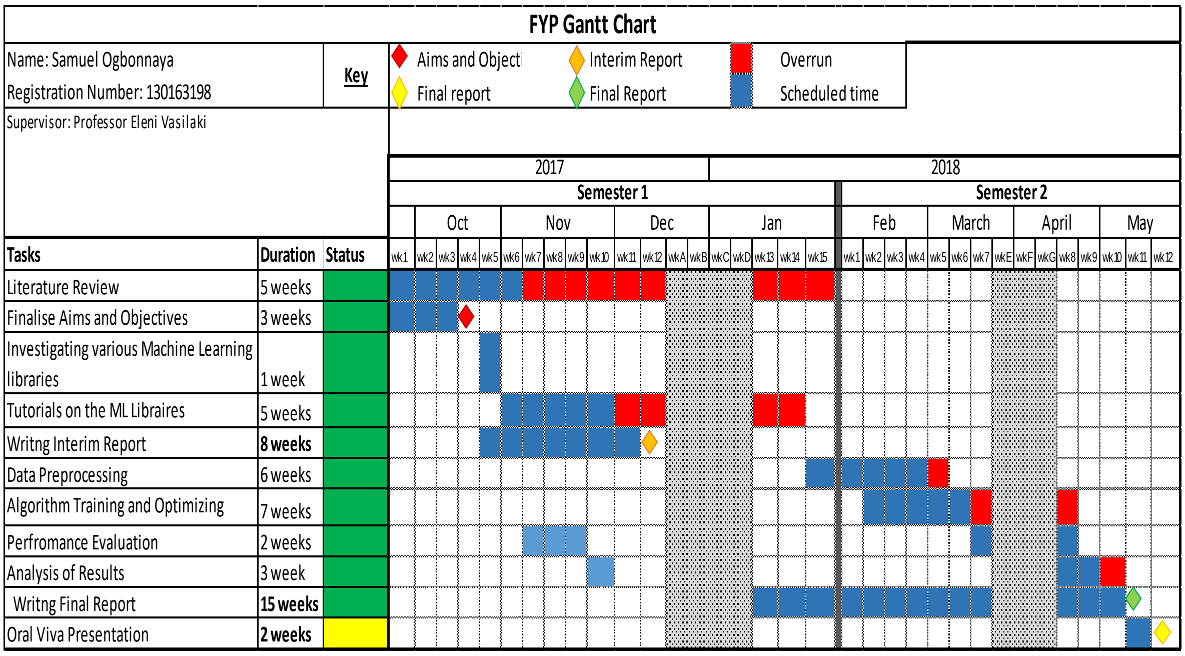

Figure 7. 1: Final Year Project Gantt Chart

8 SELF REVIEW

I believe I made good progress in terms of completing all the required tasks. By week 11, all tasks excluding the oral viva presentation were completed with 50% of the tasks being overrun due to my initial underestimation of the duration required.

By completing a comprehensive review of related literature utilising journals, textbook and other articles, I have been able to develop significant theoretical knowledge on some of the fundamental principles required for conducting a machine learning project. In addition, a huge amount of software implementation expertise was developed from this project. This was mainly through solving implementation issues such as compatibility, debugged or development of automation code to make my program more efficient. My time-management skills, self-discipline and perseverance were also further developed during this project. However, there is still room for improvement with regards to time management and planning as some task could most likely have been completed earlier.

In summary, I believe the project went very well as all the stipulated aims and objectives were completed on time. Also, taking into consideration my aerospace engineering background, I was still able to produce results which performed comparatively well with the surrounding literature.

This project overall was very worthwhile and educative. It has sparked my interest in artificial intelligence and I would hope to pursue this further in the future.