Default in Payment, an Application of Statistical Learning Techniques

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in the

DEPARTMENT OF STATISTICS

of

RHODES UNIVERSITY

by

Lulama Gcakasi

December 2019

Abstract

The ability of financial institutions to detect whether a customer will default on their credit card payment is essential for its profitability. To that effect, financial institutions have credit scoring systems in place to be able to estimate the credit risk associated with a customer. Various classification models are used to develop credit scoring systems such as k-nearest neighbours, logistic regression and classification trees. This study aims to assess the performance of different classification models on the prediction of credit card payment default. Credit data is usually of high dimension and as a result dimension reduction techniques, namely principal component analysis and linear discriminant analysis, are used in this study as a means to improve model performance. Two classification models are used, namely neural networks and support vector machines. Model performance is evaluated using accuracy and area under the curve (AUC). The neuarl network classifier performed better than the support vector machine classifier as it produced higher accuracy rates and AUC values. Dimension reduction techniques were not effective in improving model performance but did result in less computationally expensive models.

Keywords: Default in Payment, Neural Networks, Support Vector Machines, Principal Component Analysis, Linear Discriminant Analysis, Statistical Learning, Classification, Accuracy, Area Under the Curve (AUC).

Contents

List of Tables

List of Figures

List of Abbreviations

adalilne	Adaptive linear neuron.
Acc	Accuracy of a classifier.
AUC	Area under the curve.
CNS	Central nervous system.
Err	Error of a classifier.
hardlim	Hard limit function.
HLR	Hybridizing logistic regression.
LDA	Linear discriminant analysis.
logsig	Log-sigmoid function.
LOOCV	Leave out one cross validation.
MMC	Maximal margin classifier.
NN	Neural network.
ODR	Orthogonal dimension reduction.
PCA	Principal component analysis.
PNS	Peripheral nervous system.
ROC	Receiver operator curve.
SVC	Support vector classifier.
SVM	Support vector machine.

Acknowledgments

I would like to thank my supervisor Jeremy Baxter for the support, patience and understanding that he has provided throughout my time as his student. Working with Jeremy has been a pleasure and completing this thesis without his guidance and expertise would have been impossible. To my fellow masters student, Olwethu Dlangamandla, thank you for your encouragement and support. I could always rely on you to uplift me during tough days. I would like to thank Thina Maqubela for her support, mentorship and inspiring words of wisdom. I now understand that my aspirations in life are possible and achievable. The Department of Statistics is a home away from home. To Rene Zimmerman, your hard work at the department is truly appreciated. I would also like to thank my family and friends for believing in me and reminding me of my capabilities. Lastly pursuing a masters degree would not have been possible without the support and help of the Rhodes University postgraduate funding research office and for that I am forever grateful to the team.

Chapter 1

Introduction

Credit scoring is the practice of analysing a person's background information and credit application in order to predict the risk associated with awarding the person credit (Mester, 1997). Financial institutions are at risk of awarding credit to individuals who may have potential issues of non-payment or default where default is defined in this thesis as the violation of the promise to pay debt in agreed amounts and at agreed times (Islam et al., 2018). It is not easy to foresee issues such as sudden changes in a person's income due to job loss, health issues or an inability to work (Islam et al., 2018). Financial institutions have to find means to handle the risk associated with awarding credit. One way of doing this is to put an effective credit scoring system in place (Mester, 1997). A credit scoring system can be built using various statistical models including for example discriminant analysis, logistic regression and a Bayes classifier (Yeh & Lien, 2009). The purpose of chapter 5 is to develop a credit scoring system utilizing a data set generated from a bank in Taiwan which is available on the UCI Machine Learning Repository (Yeh & Lien, 2009). Neural networks, discussed in chapter 3, and support vector machines, discussed in chapter 4, are used to construct classification models to predict credit card payment default.

Van Der Maaten et al. (2009) argued that real-world data usually has high dimensionality. High dimensional data refers to data that has a large number of feature, or attribute or independent or predictor, variables and is often referred to as complex data. Several problems occur when handling high dimensional data such as multicollinearity in the context of regression. Multicollinearity occurs when two or more input variable in a regression model are closely related to each other which may result in the reduction of the accuracy of the regression coefficient estimates (James et al., 2013, pg 99) or more specifically the variance of these estimates (Radloff, 2014). In order to handle such real-world data adequately, its dimension often needs to be reduced. Scientists working with large volumes of high-dimensional data, such as global climate patterns, stellar spectra or human gene distributions, regularly confront the problem of dimensionality reduction (Tenenbaum et al., 2000). Dimensionality reduction is the transformation of high-dimensional data into a meaningful representation of reduced dimension (Van Der Maaten et al., 2009). Ideally the reduced representation has a dimension

that corresponds to the intrinsic dimensionality of the data. The intrinsic dimensionality of data is the minimum number of parameters needed to account for the observed properties of the data (Van Der Maaten et al., 2009). In simple terms the reduced representation contains as much information from the original data set as possible. Chapter 2 discusses principal components and linear discriminant analysis, the data reduction techniques utilised in the credit scoring system developed in chapter 5.

Classification, and more specifically binary classification, assessments of classifiers accuracy and model validation are discussed in this chapter. A brief summary of the thesis, including suggestions for improving the credit scoring system developed in this thesis can be found in chapter 6.

1.1 Classification

Classification is a tool used in the field of statistics to predict a qualitative response (James et al., 2013, pg 127). Suppose there exists an input vector $\mathbf{X}$ with a response $\mathbf{Y}$. If the response variable can not be quantified or belongs to a set of categories it is referred to as a qualitative response. In this context the process of modeling the relationship between $\mathbf{X}$ and $\mathbf{Y}$ is referred to as classification. A classification model or classifier is a statistical model built with the objective of classifying observations of a data set into classes (James et al., 2013, pg 127). Consider observed classes labeled $1, 2, ..., L$. The model classifies an observation as belonging to one of the L classes by drawing conclusions from the observed $\mathbf{X}$ values. For example each observation of a financial transaction data set, that is an instance of buying goods and/or using services, can be classified as "fraudulent" or "non-fraudulent". The model is able to do so by learning the relationship between the input variables and the response variable.

There are two approaches to machine learning: supervised and unsupervised learning. In supervised learning, the data set which includes the correct response variable is utilised by the model (Izenman, 2008, pg 10). This provides the model an idea of what the features resemble, for example a "fraudulent" or "non-fraudulent" transaction. The model learns this relationship and uses it to classify new observations. In unsupervised learning, the data set does not include or utilise the response variable (Izenman, 2008, pg 10). To be clear in unsupervised learning the data set may have a response variable but this variable is removed or not utilised in the learning or model estimation phase. The model searches for similarities, patterns or outliers within the data set and uses these attributes to classify new observations. This thesis focuses on supervised learning since the classifiers that will be considered will utilise the response variable in the learning phase.

A classification model is typically built by splitting a data set into a train and test data set. Denote the full data set as $\mathbf{D}$, the train set as $\mathbf{D}_{train}$ and the test set as $\mathbf{D}_{test}$. The training

set, $\mathbf{D}_{train}$, is used to build the model, that is establish parameters of the model that best describe the relationship between the dependent and independent variables of the data set (James et al., 2013, pg 15). The testing set, $\mathbf{D}_{test}$, is used to assess the performance of the model. The goal is to use the model to predict the classes of future unknown observations given the feature variables of the observations. If $L > 2$ the process of building a classifier is referred to as multi-classification. If $L = 2$ the process of building a classifier is referred to as binary classification.

1.2 Binary Classification

Binary classification is the task of classifying observations of a given data set into one of two classes on the basis of a classification rule. Suppose the Rhodes University academic office wants to predict whether students will either pass or fail at the end of the year. A data set, $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3)$, consisting of n observations on a 3-dimensional feature space and an outcome variable $\mathbf{Y} = (y_1, y_2, ..., y_n)$ of past students is available. $\mathbf{X}_1$ represents the number of hours each student spends studying per day, $\mathbf{X}_2$ represents the number of courses each student is enrolled in and $\mathbf{X}_3$ represents the intelligence quotient score of each student. $\mathbf{Y}$ is the outcome variable representing the final year mark of each student. The outcome variable can be classified into one of two groups, namely pass or fail. In other words $\mathbf{Y} \in \{pass,\ fail\}$ based on a classification rule. It is the process of classifying the observations into one of the two classes that is referred to as binary classification. The analysis chapter, chapter 5, of this thesis focuses on a binary classification problem. More formally, each multivariate observation $\mathbf{X}_i$, $i = 1, ..., n$, is mapped to one element of the set $\mathbf{Y} = \{Y_1, Y_2\}$. In binary classification problems the class of interest is typically referred to as the positive class and the other class is referred to as the negative class. When the model has been built the test data set,$\mathbf{D}_{test}$, is used to predict the classes of the observations in the set and the result is used to determine how well the model performs. There are various methods used to evaluate the performance of a classifier, see section 1.4 on the following page.

1.3 Multiclass Classification

Binary classification is the process of classifying observations into one of two groups. Multiclass classification is an extension of binary classification from two to three or more classes (Izenman, 2008, pg 260). Multiclass classification is the task of classifying observations into one of three or more classes based on a classification rule. Consider the scenario in section 1.2 but suppose the Rhodes University academic office also wants to predict whether a student will drop out of university at the end of the year. The outcome variable $\mathbf{Y}$ can now be classified into one of three groups, namely pass, fail or dropout. In other words

$\mathbf{Y} \in \{pass, fail, dropout\}$ based on a classification rule. It is the process of classifying the observations into one of the three or more classes that is referred to as multiclass classification.

1.4 Supervised Binary Classifier Performance and Evaluation

After a classifier has been developed it is important to assess its performance before it is used to classify new observations (Izenman, 2008, pg 10). Various model performance measures can be used to evaluate the prediction accuracy of a classifier, for example specificity and sensitivity. Consider a data set with n observations in p dimensional space and a binary classifier. Denote the true class of an observation from the data set as y_i and the feature vector of the observation as $\mathbf{x}_i \in \mathbb{R}^p$. Suppose a classifier, denoted as M, classifies each observation $\mathbf{x}_i$ to a class denoted by $\hat{y}_i = M(\mathbf{x}_i)$. The classifier is thus the mapping $M : \mathbf{x}_i \mapsto \hat{y}_i = f(\mathbf{x}_i)$ for some function $f(\cdot)$. Define an indicator function, denoted as I, that has value 1 if the observed class of the observation is not equal to the predicted class of the observation and has value 0 otherwise. A misclassification can be represented by the indicator function as:

$$I(y_i \neq \hat{y}_i) = \begin{cases} 1, & y_i \neq \hat{y}_i \\ 0, & y_i = \hat{y}_i \end{cases}$$

for each observation $(\mathbf{x}_i, y_i)$ and corresponding predicted class label $\hat{y}_i$. This can be used to formulate the error rate of the classifier defined as the proportion of misclassified observations (Izenman, 2008, pg 12), that is

$$\text{Error rate} = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq \hat{y}_i).$$

The best classifier is the function M where the error rate is the smallest (James et al., 2013, pg 37). Similarly a correct classification can be represented by an indicator function as:

$$I(y_i = \hat{y}_i) = \begin{cases} 1, & y_i = \hat{y}_i \\ 0, & y_i \neq \hat{y}_i \end{cases}$$

for each observation $(\mathbf{x}_i, y_i)$ and corresponding predicted class label $\hat{y}_i$. The accuracy of the classifier is defined as the proportion of correct predictions, that is

$$\text{Accuracy} = \frac{1}{n} \sum_{i=1}^{n} I(y_i = \hat{y}_i) = 1 - \text{Error rate}.$$

The higher the accuracy the better the classifier (Zaki et al., 2014, pg 602).

When a data set is used as the input to a binary classification model for prediction there are four possible outcomes. For a binary classifier suppose $\mathbf{Y}$ has outcomes one or zero where class one is the positive class. Consider an observation that belongs to the positive class, class one. If this observation is correctly predicted to belong to class one this outcome is referred to as a true positive (tp). If it is incorrectly predicted to belong to class zero this outcome is referred to as a false negative (fn). In a similar fashion consider an observation that belongs to the negative or zero class. If it is correctly predicted to belong to class zero this outcome is referred to as a true negative (tn) and if it is incorrectly predicted to belong to class one this outcome is referred to as a false positive (fp). The outcomes of the classifier applied to a data set can be represented by a two-by-two matrix, the confusion matrix, as shown in table 1.1.

Table 1.1: The confusion matrix.

	Actual		
Predicted	**Data class**	**Positive**	**Negative**
	Positive	true positives	false positives
	Negative	false negatives	true negatives

This matrix is typically referred to as the confusion matrix. This matrix is used to calculate several metrics that measure the performance of the classifier or model (table 1.2). The error rate, and hence the accuracy, is one of the most common metrics used to evaluate classification model performance (Ling et al., 2003). The error rate for a binary classifier is given as the proportion of false predictions, that is

$$\text{Error rate} = \frac{fp + fn}{n}$$

and the accuracy is given as the proportion of correct predictions,

$$\text{Accuracy} = \frac{tp + tn}{n} = 1 - \text{Error rate}.$$

These metrics are considered as global measures of classifier performance since they do not explicitly consider the classes that contribute to the error (Zaki et al., 2014, pg 603). A more class-specific performance measure is precision. Precision is defined as the proportion of correct predictions over all points predicted to be in a particular class. The precision for the positive class is the number of true positives divided by the total number of positive labels, that is $\text{prec}_p = \frac{tp}{tp+fp}$. Similarly the precision for the negative class is the number of true negatives divided by the total number of negative labels, that is $\text{prec}_n = \frac{tn}{tn+fn}$.

Additional performance measures are listed in table 1.2, for example sensitivity, the true positive rate, and specificity, the true negative rate. Sensitivity is defined as the proportion of correct positive predictions with respect to all observations actually in the positive class. Specificity is defined as the proportion of correct negative predictions with respect to all

observations actually in the negative class.

Table 1.2: Metrics derived from the confusion matrix.

Measure	Formula	Evaluation focus
Error rate	$\frac{fp+fn}{n}$	Overall error of a classifier.
Accuracy	$\frac{tp+tn}{n}$	Overall accuracy of a classifier.
Precision (prec_p)	$\frac{tp}{tp+fp}$	Class agreement of the data labels with the positive labels given by the classifier.
Sensitivity	$\frac{tp}{tp+fn}$	How effectively a classifier identifies positive labels.
Specificity	$\frac{tn}{fp+tn}$	How effectively a classifier identifies negative labels.
F score	$\frac{(\beta^2+1)tp}{(\beta^2+1)tp+\beta^2 fn+fp}$	Relations between data's positive labels and those given by a classifier.
AUC	$\frac{1}{2}\left(\frac{tp}{tp+fn}+\frac{tn}{tn+fp}\right)$	Classifier's ability to avoid false classification.

While accuracy may be a simple and useful tool to evaluate a model's performance it can sometimes provide a poor measurement of model performance especially in the case of imbalanced data (Fawcett, 2006). Imbalanced data occurs when one class is represented by a large number of observations, while the other is represented by only a few (Batista et al., 2004). Imbalanced data hinders the performance of classification techniques particularly when the minority class is the positive class (Chawla et al., 2004). As a result alternative methods to measure model performance should be considered. An example of such a method is a receiver operating characteristics (ROC) graph which is a popular strategy for visualising, organising and selecting classifiers based on their performance (Fawcett, 2006). A ROC graph is a two-dimensional graph in which the true positive rate is plotted on the vertical or y-axis and the false positive rate is plotted on the horizontal or x-axis, see figure 1.1. This graph demonstrates the relative tradeoffs between the benefits and costs of a classifier (Fawcett, 2006). It is a popular strategy for assessing model performance in binary classification (Zaki et al., 2014, pg 610). ROC graphs have properties that make them especially useful for classifications problems with imbalanced data sets (Fawcett, 2006).

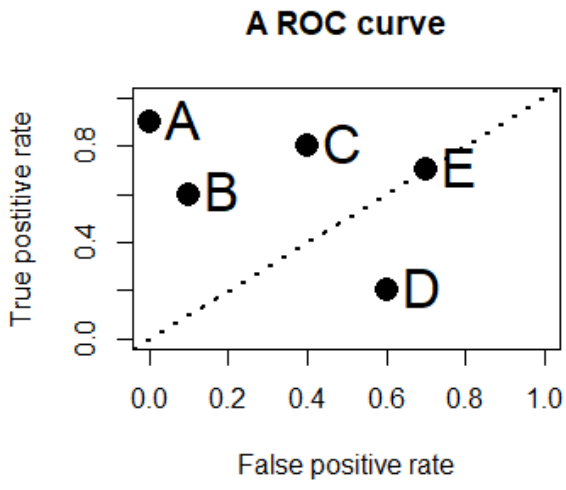


Figure 1.1: A ROC graph showing five classifiers (Fawcett, 2006).

A binary classifier outputs a class label for each feature vector $\mathbf{x}_i$. Binary classifiers produce a single point on the ROC space. Each binary classifier produces a false positive rate, true positive rate pair for all feature vectors in the data set, typically the test data set. This pair corresponds to a single point in ROC space. Consider five binary classifiers, denoted by A, ..., E, which produce false positive and true positive rate pairs corresponding to a single point in the ROC space as shown in figure 1.1. The classifier at point (0,0) assigns no observations to the positive class which results in 0% true positive and false positive rates. The classifier at point (1, 1) assigns all observation to the positive class which results in a 100% true positive and false positive rates. A perfect classifier lies at the point (0, 1) where the false positive rate is 0% and the true positive rate is 100%. Generally a good classifier lies at the point located in the upper left-hand region of the ROC graph where the true positive rate is maximal and the false positive rate is minimal. In figure 1.1, classifier A may be considered as the best classifier since it has the lowest false positive rate and the highest true positive rate out of all the classifiers. The diagonal line connecting the (0, 0) and (1, 1) coordinates is called the "chance diagonal" and represents the strategy of randomly guessing the class of an observation, which is not a good classification strategy. In figure 1.1, classifier E uses this strategy as it guesses the positive class 70% of the time and can therefore be expected to get a 70% true positive rate and a 70% false positive rate. Classifiers with points on the lower left-hand side of the graph are said to be conservative as they minimise false positive predictions. These classifiers typically have a low true positive rate (Fawcett, 2006). Classifiers with points on the upper right-hand side of the graph are said to be liberal as they mostly make correct positive predictions but with a high false positive rate (Fawcett, 2006). In figure 1.1, classifier B is more conservative than classifier C. Classifier C is more liberal than classifier B. Classifiers on the lower right-hand of the graph are not considered as good classifiers as they perform worse than classifiers that use random guessing, that is the classifiers on the chance diagonal, as a strategy to assign observations.

Some classifiers, for example neural networks, produce a numerical value or score, for example a probability, as a response variable which can be used to decide which class an observation should be assigned to. In these instances, the classifier chooses some positive score threshold, ρ, and assigns all observations with score above ρ to the positive class while the remaining points are classified as negative. Such a threshold is likely to be arbitrary. ROC analysis plots the performance of the classifier over all possible values of the threshold parameter tracing a curve through ROC space (Zaki et al., 2014, 610). Figure 1.2 (Fawcett, 2006) shows an example of a ROC graph on a test set of 20 observations. A ROC curve generated from a finite set of observations produces a step function (Fawcett, 2006). As the number of observations approaches infinity, the step function begins to resemble a more curve-like structure (Fawcett, 2006). In figure 1.2 each observation produces a point in ROC space which has a score threshold used to produce it. Varying the threshold value produces different points in ROC space. For these data, a threshold of positive infinity produces point (0, 0). A threshold of 0.9 gives the first observation classified as positive with point (0, 0.1). As the threshold is

lowered the shape of the curve takes form and starts to resemble a logarithmic curve. The curve stops at point (1, 1) with threshold 0.1. At the point (0.1, 0.5), with a threshold of 0.54, the classifier produces its highest accuracy. This means a good score threshold for this classifier is 0.54 rather than 0.5.

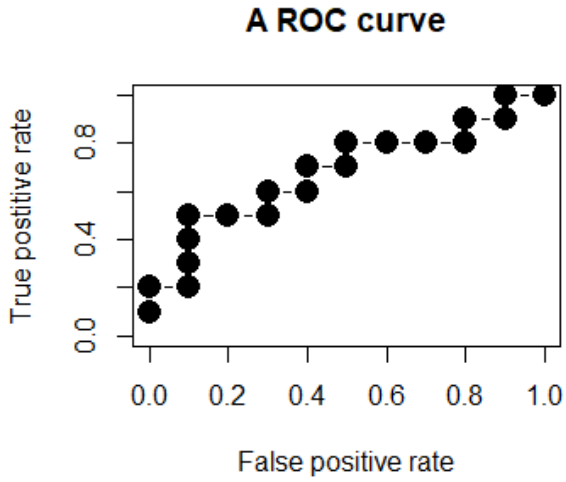


Figure 1.2: An example of a ROC curve for $n = 20$ observation (Fawcett, 2006).

The area under the ROC curve, denoted as AUC, can be used as a measure of classifier performance. Since the total area of the plot is 1, the AUC lies in the interval [0,1] and the higher the AUC the better the performance. The AUC value is essentially the probability that the classifier will rank a random positive test instance higher than a random negative test instance (Fawcett, 2006). AUC indicates how much a model is capable of distinguishing between classes. A perfect model has an AUC of one (1) which indicates an excellent measure of separability. The closer the AUC to 1 the better the model. When the AUC is 0.8 it means there is an 80% chance that model will be able to distinguish between positive class and negative class. An AUC of 0.5 indicates a model that has no capacity to distinguish between positive class and negative class. This is the case of a model that uses the strategy of randomly guessing the class of an observation to assign the observations into the positive or negative class. When the AUC is 0 the model reciprocates the classes. This means the model is assigning positive observations to the negative class and negative observations to the positive class. The AUC makes it easy to compare the ROC curve of one model to another and hence compare the performance of a model to another (Zaki et al., 2014, pg 611).

1.5 Model Validation

Model validation is defined as the process of determining the extent to which a model's predictions resembles real world outcomes as a means to decide if the model is suitable for its intended purpose (Mayer & Butler, 1993). There are various model validation techniques used across different fields of study. Choosing a model validation technique depends on the complexities of the model in question and the type of data used to build the model (Mayer & Butler, 1993). Cross validation is a common approach to model validation (Zaki et al., 2014, pg 616).

1.5.1 Cross Validation

Cross validation is a method of assessing and comparing classifiers by dividing the data set, $\mathbf{D}$, into a training set, denoted as $\mathbf{D}_{train}$, and a validation or test set, denoted as $\mathbf{D}_v$ (James et al., 2013, pg 176). The two disjoint sets are independent and have the same distribution as the original data set. The training set is used to build the model and the validation set is used to evaluate the suitability of the model. An important trait of cross validation is that each observation in the data set has an equal opportunity of being selected to be part of either the training set or the validation set. In classification, the model accuracy and error rate can be calculated using the validation set and the result can be used for model comparison and selection. Denote the accuracy calculated using the validation set as Acc_v and the error rate calculated using the validation set as Err_v. The following sections discuss different approaches to cross validation such as k-fold cross validation.

1.5.1.1 k-Fold Cross Validation

In k-fold cross validation, the data set is randomly partitioned into k disjoint equal sized sets called folds (James et al., 2013, pg 181). One fold is used as the validation set and the remaining folds are used to train or build the model (Zaki et al., 2014, pg 616). The process is repeated k times such that all the folds are used as the validation set. Denote the folds as $\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_k$ and the binary classifier built from $\mathbf{D} \setminus \mathbf{D}_j$ as M_j, $j = 1, 2, \cdots, k$ where $\mathbf{D} \setminus \mathbf{D}_j$ denotes the data set $\mathbf{D}$ with the observations in $\mathbf{D}_j$ removed. For each iteration classifier M_j is built using $\mathbf{D} \setminus \mathbf{D}_j$ and evaluated using the validation set $\mathbf{D}_j$. The accuracy in fold j is denoted as Acc_v^j where $Acc_v^j = \frac{1}{n_j} \sum_{i=1}^{n_j} I\left(y_i = \hat{y}_i, \mathbf{X}_i \in \mathbf{D}_j\right)$ and the error rate is given by $Err_v^j = \frac{1}{n_j} \sum_{i=1}^{n_j} I\left(y_i \neq \hat{y}_i, \mathbf{X}_i \in \mathbf{D}_j\right)$, calculated over $\mathbf{D}_j$. The overall accuracy of the model using the k-fold cross validation method is determined by averaging the accuracy rates

resulting from the k iterations and is given as

$$Acc_v = \frac{1}{k}\sum_{j=1}^{k} Acc_v^j = \frac{1}{k}\sum_{j=1}^{k}\left[\frac{1}{n_j}\sum_{i=1}^{n_j} I\left(y_i = \hat{y}_i\right)\right]$$

where $\mathbf{X}_i \in \mathbf{D}_j$ and $n_j = \mid \mathbf{D}_j \mid$ that is the number of observations in $\mathbf{D}_j$. Similarly the overall error rate of the model using the k-fold cross validation method is determined by averaging the error rates resulting from the k iterations and is given as

$$Err_v = \frac{1}{k}\sum_{j=1}^{k} Err_v^j = \frac{1}{k}\sum_{j=1}^{k}\left[\frac{1}{n_j}\sum_{i=1}^{n_j} I\left(y_i \neq \hat{y}_i\right)\right]$$

where $\mathbf{X}_i \in \mathbf{D}_j$ and $n_j = \mid \mathbf{D}_j \mid$. Consider a 10-fold cross validation example. The data set is partitioned into 10 folds, denoted as $\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_{10}$. In the first iteration the training data, $\mathbf{D}_{train}$, consists of all the folds but $\mathbf{D}_1$, that is $\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_1$. The accuracy, Acc_v^1, and error rate, Err_v^1, are calculated using the validation set $\mathbf{D}_1$. In the second iteration the training data consists of all the folds but $\mathbf{D}_2$, that is $\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_2$. The accuracy, Acc_v^2, and error rate, Err_v^2, are calculated using the validation set $\mathbf{D}_2$. The process continues for all 10 folds. Table 1.3 shows the complete process. The accuracy and error rate are given as

$$Acc_v = \frac{1}{10}\sum_{j=1}^{10} Acc_v^j \text{ and } Err_v = \frac{1}{10}\sum_{j=1}^{10} Err_v^j.$$

Typically k is chosen to be between 5 and 10 (Zaki et al., 2014, pg 616). The case where $k = n$ is referred to as leave-one-out cross validation (James et al., 2013, pg 178).

Table 1.3: The 10-fold cross validation process.

Iteration	Training set	Validation set	Accuracy	Error rate
1	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_1$	$\mathbf{D}_1$	Acc_v^1	Err_v^1
2	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_2$	$\mathbf{D}_2$	Acc_v^2	Err_v^2
3	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_3$	$\mathbf{D}_3$	Acc_v^3	Err_v^3
4	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_4$	$\mathbf{D}_4$	Acc_v^4	Err_v^4
5	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_5$	$\mathbf{D}_5$	Acc_v^5	Err_v^5
6	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_6$	$\mathbf{D}_6$	Acc_v^6	Err_v^6
7	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_7$	$\mathbf{D}_7$	Acc_v^7	Err_v^7
8	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_8$	$\mathbf{D}_8$	Acc_v^8	Err_v^8
9	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_9$	$\mathbf{D}_9$	Acc_v^9	Err_v^9
10	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_{10}$	$\mathbf{D}_{10}$	Acc_v^{10}	Err_v^{10}

1.5.1.2 Leave-One-Out Cross Validation

In leave-one-out cross validation, abbreviated LOOCV, the data set is partitioned into n sets where n is the number of observations (James et al., 2013, pg 178). This results in each

set being a single observation. Table 1.4 summarises the LOOCV process. One set, namely $\mathbf{D}_j = (\mathbf{x}_j, y_j)$, is used as the validation set while the remaining sets are used to train or build the model, that is $\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_j = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_n, y_n) \setminus (\mathbf{x}_j, y_j)\}$. The process is repeated n times such that each observation is used only once as the validation set. In each iteration the classifier is evaluated using a single observation $\mathbf{D}_j = (\mathbf{x}_j, y_j)$. The accuracy and error rate of the classifier for fold j is given by $Acc_v^j = I(y_i = \hat{y}_i,\ \mathbf{x}_i \in \mathbf{D}_j)$ and $Err_v^j = I(y_i \neq \hat{y}_i,\ \mathbf{x}_i \in \mathbf{D}_j)$ respectively. The overall accuracy of the classifier is given by

$$Acc_v = \frac{1}{n}\sum_{j=1}^{n} Acc_v^j = \frac{1}{n}\sum_{j=1}^{n} [I(y_i = \hat{y}_i)]$$

and the overall error rate of the classifier is given by

$$Err_v = \frac{1}{n}\sum_{j=1}^{n} Err_v^j = \frac{1}{n}\sum_{j=1}^{n} [I(y_i \neq \hat{y}_i)]$$

where $x_i \in \mathbf{D}_j$.

Table 1.4: The LOOCV cross validation process.

Iteration	Training set	Validation set	Accuracy	Error rate
1	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_1$	$\mathbf{D}_1 = (\mathbf{x}_1, y_1)$	Acc_v^1	Err_v^1
2	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_2$	$\mathbf{D}_2 = (\mathbf{x}_2, y_2)$	Acc_v^2	Err_v^2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n-1$	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_{n-1}$	$\mathbf{D}_{n-1} = (\mathbf{x}_{n-1}, y_{n-1})$	Acc_v^{n-1}	Err_v^{n-1}
n	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_n$	$\mathbf{D}_n = (\mathbf{x}_n, y_n)$	Acc_v^n	Err_v^n

1.5.1.3 Hold-Out Cross Validation

Hold-out cross validation is a special case of k-fold cross validation where $k = 2$ (Kohavi, 1995). Table 1.5 summarises the hold-out cross validation process. The data set is partitioned into two disjoint sets denoted $\mathbf{D}_1$ and $\mathbf{D}_2$. One set is used to train or build the model, that is $\mathbf{D}_{train} = \mathbf{D}_1$, while the second set, $\mathbf{D}_2$, is used to evaluate the model. The process is repeated once more with the training set changing to $\mathbf{D}_{train} = \mathbf{D}_2$ and the remaining set $\mathbf{D}_1$ is used as the validation set. The procedure has two iterations such that each set is used only once as the validation set. The accuracy and error rate for each iteration is given by $Acc_v^j = \frac{1}{n_j}\sum_{i=1}^{n_j} I(y_i = \hat{y}_i)$ and $Err_v^j = \frac{1}{n_j}\sum_{i=1}^{n_j} I(y_i \neq \hat{y}_i)$ respectively where $\mathbf{X}_i \in \mathbf{D}_j$ and $n_j = \mid \mathbf{D}_j \mid$. The overall accuracy of the classifier is given by

$$Acc_v = \frac{1}{2}\sum_{j=1}^{2} Acc_v^j = \frac{1}{2}\sum_{j=1}^{2} [I(y_i = \hat{y}_i)]$$

and the overall error rate of the classifier is given by

$$Err_v = \frac{1}{2}\sum_{j=1}^{2} Err_v^j = \frac{1}{2}\sum_{j=1}^{2} \left[I\left(y_i \neq \hat{y}_i\right)\right]$$

where $\mathbf{X}_i \in \mathbf{D}_j$ and $n_j = \mid \mathbf{D}_j \mid$.

Table 1.5: The hold-out cross validation process.

Iteration	Training set	Validation set	Accuracy	Error rate
1	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_1 = \mathbf{D}_2$	$\mathbf{D}_1$	Acc_v^1	Err_v^1
2	$\mathbf{D}_{train} = \mathbf{D} \setminus \mathbf{D}_2 = \mathbf{D}_1$	$\mathbf{D}_2$	Acc_v^2	Err_v^2

1.5.2 The Bootstrap

Another approach to model validation is bootstrap resampling. Instead of dividing the data set into exclusive sets or folds, the bootstrap method selects B random samples of size n with replacement (Hastie et al., 2001, pg 217). This results in B sets, denoted as $D_1, D_2, \cdots, D_B$, with several repeated points. Table 1.6 provides a summary of the bootstrap validation approach. Note that the sets are the same size as the original data set. Each sample, D_j, is used to train or build the model and the resulting model is evaluated using the original data set (Zaki et al., 2014, pg 618). This process is repeated B times such that each set is used once as the training set. Each iteration produces a classifier M_j, $j = 1, 2, \cdots, B$, that is subsequently used to calculate the accuracy, $Acc_B^j = \frac{1}{n}\sum_{i=1}^{n} I\left(y_i = \hat{y}_i\right)$, and error rate, $Err_B^j = \frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{y}_i\right)$, of the classifier using the original data. The overall accuracy of the model using the bootstrap method is determined by averaging the accuracy rates resulting from the B iterations and is given as

$$Acc_B = \frac{1}{B}\sum_{j=1}^{B} Acc_B^j = \frac{1}{B}\sum_{j=1}^{B}\left[\frac{1}{n}\sum_{i=1}^{n} I\left(y_i = \hat{y}_i\right)\right]$$

where $\mathbf{X}_i \in \mathbf{D}$. Similarly the overall error rate of the model using the bootstrap method is determined by averaging the error rates resulting from the B iterations and is given as

$$Err_B = \frac{1}{B}\sum_{j=1}^{B} Err_B^j = \frac{1}{B}\sum_{j=1}^{B}\left[\frac{1}{n}\sum_{i=1}^{n} I\left(y_i \neq \hat{y}_i\right)\right]$$

where $\mathbf{X}_i \in \mathbf{D}$.

Table 1.6: The bootstrap validation process.

Iteration	Training set	Validation set	Accuracy	Error rate
1	$\mathbf{D}_{train} = \mathbf{D}_1$	$\mathbf{D}$	Acc_B^1	Err_B^1
2	$\mathbf{D}_{train} = \mathbf{D}_2$	$\mathbf{D}$	Acc_B^2	Err_B^2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
B	$\mathbf{D}_{train} = \mathbf{D}_B$	$\mathbf{D}$	Acc_B^B	Err_B^B

1.5.3 Choice of Model Validation Methodology.

The drawback of using cross validation to estimate the test error is that it can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set (James et al., 2013, pg 178). In this approach only a subset of the observations, those that are included in the training set rather than in the validation set, are used to fit the model. This suggests that the cross validation set error may tend to overestimate the test error for the model fit on the entire data set (James et al., 2013, pg 178). On the other hand bootstrapping tends to drastically reduce the variance but gives more biased results (Hastie et al., 2001, pg 218). In k-fold cross validation each of the k validation folds is distinct from the other $k-1$ folds used for training, that is there is no overlap. This plays a part in its success. To estimate prediction error using the bootstrap the same procedure used in cross validation is considered, that is using each bootstrap data set as the training sample and the original sample as the validation sample. The problem is each bootstrap sample has significant overlap with the original data (Hastie et al., 2001, pg 218). Approximately two-thirds of the original data points appear in each bootstrap sample (Zaki et al., 2014, 618). This causes the bootstrap to seriously underestimate the true prediction error (Kohavi, 1995). One way to solve this problem is to only consider predictions for those observations that did not occur in the current bootstrap sample. Unfortunately this method gets complicated, and k-fold cross validation provides a simpler, more attractive approach for estimating prediction error (Zaki et al., 2014, pg 618).

Chapter 2

Principal Components and Linear Discriminant Analysis

Van Der Maaten et al. (2009) argued that real-world data usually has high dimensionality, that is data which has a large number of feature, or attribute or independent or predictor, variables. High dimensionality can lead to issues when fitting statistical models, for example multicollinearity in a regression model. Dimensionality reduction is the transformation of high-dimensional data into a meaningful representation of the data in a reduced dimension. Ideally the reduced representation has a dimension that corresponds to the intrinsic dimensionality of the data. The intrinsic dimensionality of data is the minimum number of parameters needed to account for the observed properties of the data (Van Der Maaten et al., 2009). The reduced representation contains as much information from the original data set as possible. Dimension reduction is important in data analysis, since it mitigates the curse of dimensionality and other undesired properties of high-dimensional data (Van Der Maaten et al., 2009). The term "curse of dimensionality" describes the difficulty of dealing with statistical problems in high dimensions (Verleysen & François, 2005).

Suppose n observations of p feature, or independent or predictor, variables are represented by $\mathbf{X}$, that is

$$\underset{n\times p}{\mathbf{X}} = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1p} \\ x_{21} & x_{22} & ... & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & ... & x_{np} \end{bmatrix}.$$

The response or dependent variable, $\mathbf{Y} = (y_1, y_2, ..., y_n)'$, is available and used to construct the supervised classifier. The goal is to use $\mathbf{X}$ to predict $\mathbf{Y}$. Linear regression is a commonly used statistical method for predicting a quantitative response when given a set of quantitative predictor variables where least squares is the typical approach to fit this model (Radloff, 2014). James et al. (2013, pg 59) reviews some of the key ideas underlying the linear regression model as well as the least squares approach that is commonly used to fit this model. Linear

regression is a statistical learning method that is widely used (Hair et al., 1998, pg 141). However linear regression has its disadvantages, for example:

- If n is not much larger than p then there can be a lot of variability in the least squares fit, resulting in over fitting and consequently poor predictions on future observations not used in model training (James et al., 2013, pg 203).
- Often some of the predictor variables used in multiple regression are not associated with the response especially when p is substantially large. Including such variables results in complex models that are difficult to interpret (James et al., 2013, pg 203).

If the number of predictor variables can be reduced without losing information and a model fitted to predict the response on the reduced set of predictor variables then we may obtain a model that is easily interpretable, provides accurate predictions and has substantially reduced variability. One way of accomplishing this is by using the principal component analysis technique or the linear discriminant analysis technique as discussed in sections 2.2 and 2.3. James et al. (2013, pg 228) provides a summary of alternative dimension reduction methods. The well known wine data set, as discussed in the following section, is used in section 2.5 on page 26 to demonstrate these techniques.

2.1 The Wine Data Set

Wine is a popular beverage, enjoyed by a broad-range of consumers across the world (Cortez et al., 2009). In Italy, wine is by far the predominant alcoholic beverage consumed (Franceschi et al., 1990). In 1991 the wine data set was uploaded into the UCI repository (Cortez et al., 2009). The data in the data set are the results of an analysis of wines grown in the same region in Italy but coming from three different cultivars. A chemical analysis determined the quantities of 13 physiochemical attributes found in each of the three types of wines for 178 observations (Polat, 2012). The result of the analysis is a data set with 178 observations, 13 feature variables and 3 classes. The response, followed by the feature variables are as follows

- Type: The type of wine: one of three classes;
- Alcohol: The quantity of alcohol in the wine;
- Malic: The quantity of malic acid in the wine;
- Ash: The quantity of ash in the wine;
- Alcalinity: Alcalinity of ash in the wine;
- Magnesium: Quantity of magnesium in the wine;

- Phenols: Total phenols in the wine;
- Flavanoids: The quantity of flavanoids in the wine;
- Nonflavanoids: The quantity of nonflavanoid phenols in the wine;
- Proanthocyanins: The quantity of proanthocyanins in the wine;
- Colour: Colour intensity of the wine;
- Hue: The quantity of hue in the wine;
- Dilution: D280/OD315 of diluted wines;
- Proline: The quantity of proline in the wine.

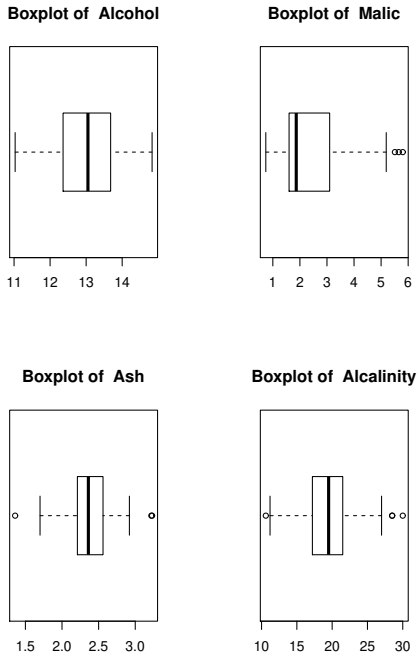
Table A.1 available in appendix A.1 on page 101 shows the first ten observations of the data set. The distribution of the classes is shown in table 2.1. This indicates that there does not exist much of a class imbalance and hence the use of metrics such as accuracy are reliable measures of classifier performance.

Table 2.1: Class distribution of the wine data set.

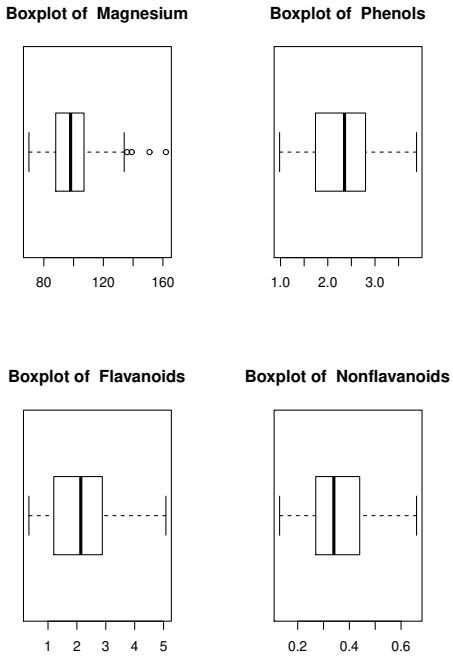
Class (label)	Frequency	Percentage
One	59	33.14607
Two	71	39.88764
Three	48	26.96629

All of the feature variables are numeric. A summary of all variables is available in table A.2 in appendix A.2 on page 101. Boxplots of each of the feature variables are shown in figures 2.1a, 2.1b, 2.1c, 2.1d on the following page. Seven of the thirteen feature variables had between one and four univariate outliers. Outliers may affect the training or learning phase when constructing a classifier. However since this is a small data set, and for the sake of simplicity, no outliers were removed. This follows the approach taken by Cortez et al. (2009).

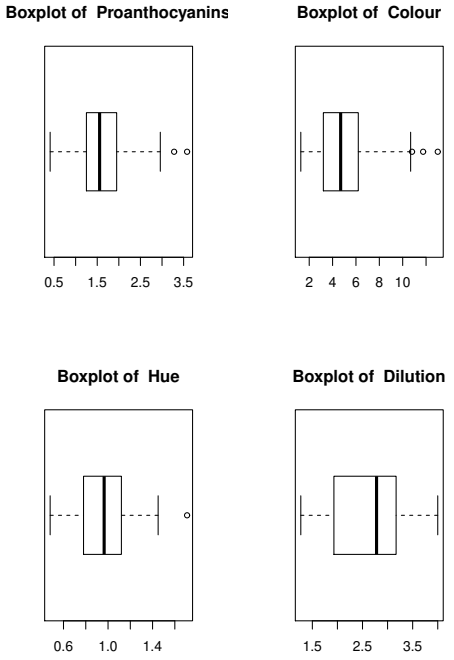
Figure 2.2a shows how the first two feature variables, alcohol and malic acid, affect the class distribution of the data. Figure 2.2a shows that there is some separability between the classes. Figure 2.2b on page 19 shows the bivariate scatterplots for all the feature variables, where type 1 is displayed in red, type 2 in blue and type 3 in green. Given the groupings of the wine types across these variables classification may be successful.



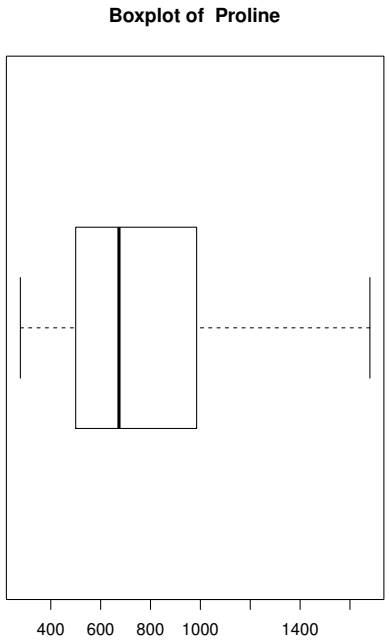
(a) Individual box plots of the first four feature variables.



(b) Individual box plots of the second four feature variables.

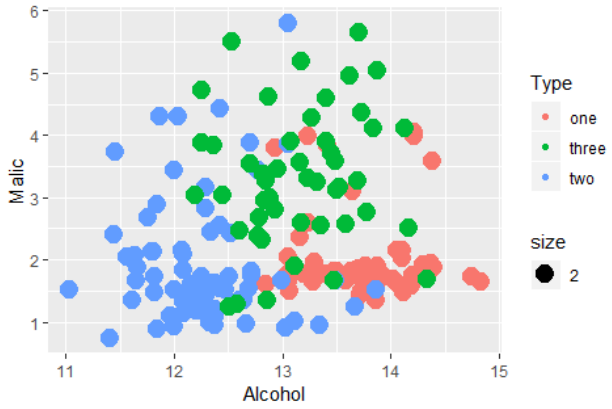


(c) Individual box plots of the third four feature variables.

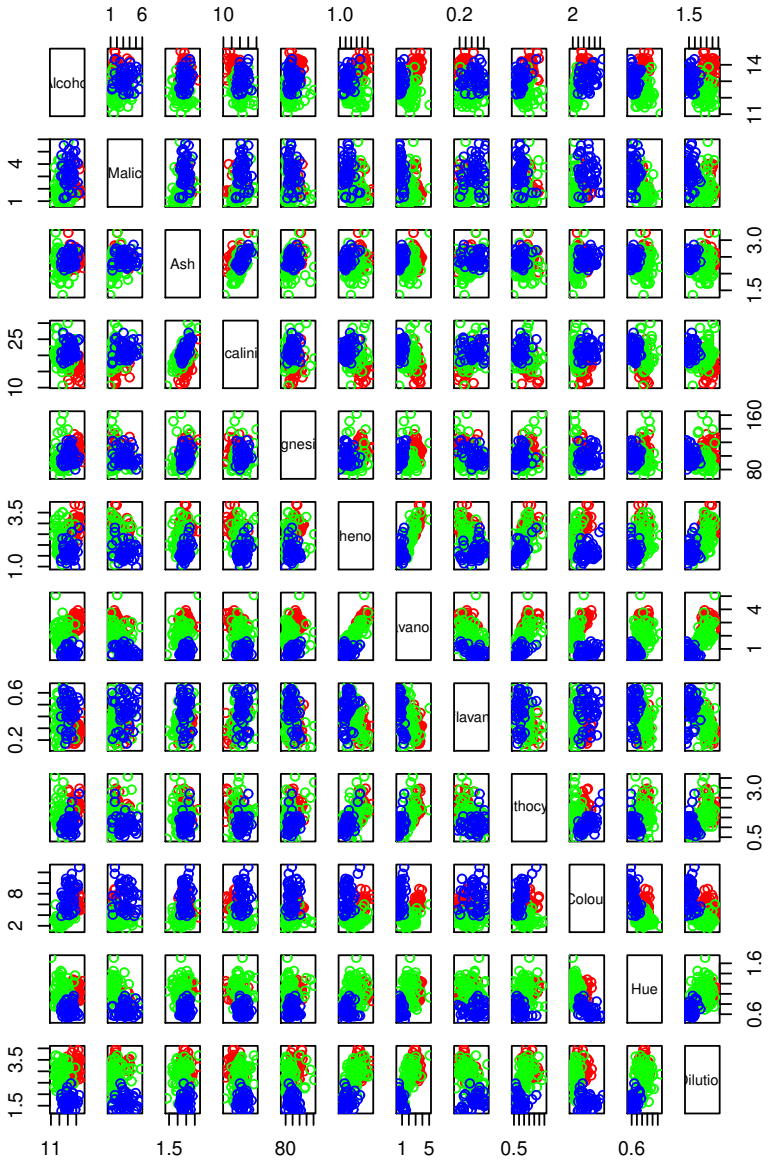


(d) Individual box plot of the last feature variables.

Figure 2.1: Boxplots of the feature variables in the wine data set.



(a) Class distribution resulting from the alcohol and malic acid feature variables.



(b) Scatterplot of the feature variables in the wine data.

Figure 2.2: Scatterplots of the wine feature variables.

This data set has thirteen (13) feature variables and may be considered as high dimensional data. Dimension reduction techniques may be considered in order to successfully perform statistical analysis on the reduced data. Sometimes data sets have highly correlated variables which might hinder the training phase. Figure 2.3 shows the correlation of the feature variables in the data set. The flavanoids feature variable is strongly, positively correlated to the phenols feature variable and semi-strongly negatively correlated to the nonflavanoids feature variable. Although there are other feature variables that have a strong correlation, not much correlation exists between most of the feature variables. This does not mean the existing correlation should be ignored. Dimension reduction also serves as a means to deal with a strong existing correlation between feature variables in a data set to improve model development. The variance-covariance matrix of these data set is available in table A.3 in appendix A.3 on page 103.

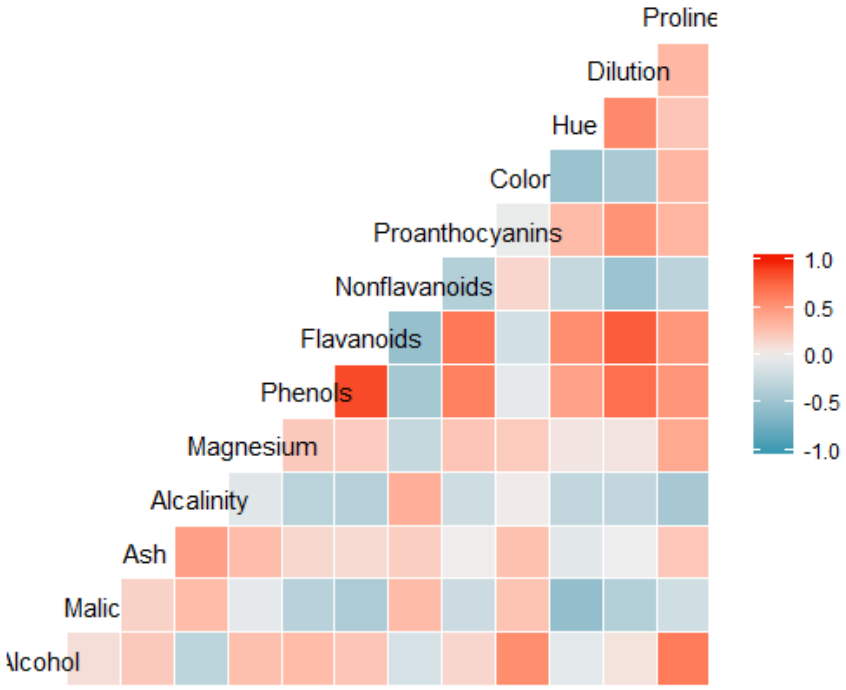


Figure 2.3: Variance-covariance diagram of the wine data feature variables.

The following sections discuss PCA and LDA and how these techniques can be applied to the wine data set. The R code is provided in appendix A.5 on page 104.

2.2 Principal Component Analysis

Principal component analysis (PCA) is a dimension reduction technique that can be used to reduce a large set of correlated predictor variables to a small set of uncorrelated variables that still contains most of the information contained in the large set (Johnson & Wichern, 1992, pg 357). Consider the n observations on a p-dimensional space where not all the

feature variables are of equal importance. The goal is to reduce the original dimensional space to one that consists of sufficient dimensions that explain as much of the variation in the original set as possible, that is the reduced data set retains all or the majority of the information contained in the original dimensional space. The reduced dimensions are called principal components where each of the principal components are a linear combination of the p-predictor variables. The general aim of principal component analysis is that the first few components will account for a substantial proportion of the variation in the original predictor variables and can consequently be used to provide a convenient lower-dimensional summary of these variables that might prove useful for a variety of reasons (Everitt & Hothorn, 2006, pg 215).

The first principal component, denoted by $\mathbf{Z}_1$, is the normalised linear combination of the predictor variables that has the largest variance. The second principal component, $\mathbf{Z}_2$, is the normalised linear combination of the predictor variables that has the largest variance out of all the linear combinations uncorrelated with $\mathbf{Z}_1$. In general the j^{th} principal component, $\mathbf{Z}_j$, is the normalised linear combination of the predictor variables that has the largest variance out of all the linear combinations uncorrelated with the existing $j-1$ principal components. The method continues until a set of principal components are derived. Algorithm 2.1 describes how the principal components are derived.

Algorithm 2.1 Derivation of Principal Components:

1. Construct $\mathbf{X}$, the data matrix.
2. Standardise $\mathbf{X}$ by subtracting the mean of each column from the entries in the column. Call the resulting vector by $\mathbf{X}_{std}$.
3. Compute $\underset{p\times p}{\mathbf{S}_{n-1}}$, an estimate of the covariance matrix, $\mathbf{\Sigma}_{p\times p}$, of $\mathbf{X}_{std}$.
4. Compute the eigenvalues, denoted by $\lambda_1, \lambda_2, ..., \lambda_p$, of $\mathbf{S}_{n-1}$.
5. Compute $\underset{p\times p}{\mathbf{U}}$, the matrix whose columns are the orthonormal eigenvectors of $\mathbf{S}_{n-1}$, sorted by decreasing eigenvalues.
6. Choose $k < p$ and construct $\underset{p\times k}{\mathbf{U}^*}$, the matrix consisting of the first k columns of $\mathbf{U}$.
7. Compute $\underset{n\times k}{\mathbf{Z}}$, where $\underset{n\times k}{\mathbf{Z}} = \mathbf{X}\mathbf{U}^*$.

$\mathbf{Z}$ is a reduced dimensional space representation of $\mathbf{X}$ that may contain most of the information contained in $\mathbf{X}$. We can use $\mathbf{Z}$ to fit a linear regression model using least squares. This is referred to as principal component regression. James et al. (2013, pg 230) gives a brief description of the principal component regression approach.

A very important consideration is the choice k, the number of principal components. If most, for example 80% to 90%, of the total variance for large p can be attributed to the first one,

two or three principal components then the original p feature variables can be replaced by these principal components without much loss of information (Johnson & Wichern, 1992, pg 359). Jolliffe (2002) provides a summary of existing approaches to determining the number principal components.

Example 2.1. A data set, adapted from Tharwat (2016), consisting of six observations with three feature variables is represented by $\mathbf{X}$,

$$\mathbf{X} = \begin{bmatrix} 4 & 21 & 40 \\ 3 & 19 & 59 \\ 5 & 19 & 55 \\ 5 & 20 & 61 \\ 7 & 22 & 43 \\ 6 & 22 & 42 \end{bmatrix}.$$

The mean vector of $\mathbf{X}$ is $\bar{\mathbf{x}} = \begin{bmatrix} 5 & 20.25 & 50 \end{bmatrix}'$ and after standardising, we obtain

$$\mathbf{X}_{std} = \begin{bmatrix} -1 & 0.5 & -10 \\ -2 & -1.5 & 9 \\ 0 & -1.5 & 5 \\ 0 & -0.5 & 11 \\ 2 & 1.5 & -7 \\ 1 & 1.5 & -8 \end{bmatrix}.$$

The covariance matrix of $\mathbf{X}_{std}$ is given by

$$\mathbf{S}_{n-1} = \begin{bmatrix} 2 & 1.4 & -6 \\ 1.4 & 1.9 & -10.8 \\ -6 & -10.8 & 88 \end{bmatrix}.$$

The eigenvalues of $\mathbf{S}_{n-1}$ are $\lambda_1 = 89.7617204$, $\lambda_2 = 1.8929842$ and $\lambda_3 = 0.2452955$. The corresponding, that is in the same order, matrix of orthonormal eigenvectors of $\mathbf{S}_{n-1}$ is given by

$$\mathbf{U} = \begin{bmatrix} -0.06964117 & 0.8952440 & -0.44010041 \\ -0.12279906 & 0.4301229 & 0.89437950 \\ 0.98998510 & 0.1163296 & 0.07998085 \end{bmatrix}.$$

The columns of $\mathbf{U}$ are the corresponding eigenvectors sorted by decreasing eigenvalues. Let $k = 2 < p = 3$, then

$$\mathbf{U}^* = \begin{bmatrix} -0.06964117 & 0.8952440 \\ -0.12279906 & 0.4301229 \\ 0.98998510 & 0.1163296 \end{bmatrix}.$$

The new sample space is given by

$$\mathbf{Z} = \mathbf{X}\mathbf{U}^* = \begin{bmatrix} 4 & 21 & 40 \\ 3 & 19 & 59 \\ 5 & 19 & 55 \\ 5 & 20 & 61 \\ 7 & 22 & 43 \\ 6 & 22 & 42 \end{bmatrix} \begin{bmatrix} -0.06964117 & 0.8952440 \\ -0.12279906 & 0.4301229 \\ 0.98998510 & 0.1163296 \end{bmatrix} = \begin{bmatrix} 36.74206 & 17.26674 \\ 55.86702 & 17.72151 \\ 51.76779 & 19.04668 \\ 57.58490 & 20.17478 \\ 39.38029 & 20.73158 \\ 38.45995 & 19.72001 \end{bmatrix}$$

which is a 6×2 matrix and is smaller than the original 6×3 matrix.

Now consider the same data matrix, $\mathbf{X}$, but with a qualitative response $\mathbf{Y}$, where $\mathbf{Y}$ is in the form of class labels. The goal is to predict $\mathbf{Y}$. Using the linear regression model is not ideal since it assumes that the response $\mathbf{Y}$ is quantitative. Reducing the dimensional space of $\mathbf{X}$ may be necessary to avoid obtaining complex models that are difficult to interpret. A typical solution to this problem is classification. One example of a classification technique is linear discriminant analysis as discussed in section 2.3. Chapters 3 and 4 discuss neural networks and support vector machines. James et al. (2013, 127) provides a summary of alternative classification techniques.

2.3 Linear Discriminant Analysis

Linear discriminant analysis (LDA) was proposed by R. Fischer in 1936 (Fisher, 1936). It is a dimension reduction technique used to find a linear combination of features that characterises or separates two or more classes of objects or events (Li & Wang, 2014). Consider the n observations on a p-dimensional feature space where not all the dimensions are relevant. The idea of LDA is to project the feature space onto a smaller subspace while maintaining the class-discriminatory information. Suppose each observation belongs to one of L classes labeled $1, 2, ..., L$. Each class has n_i observations or samples where $i = 1, 2, \ldots, L$. The observations from the different classes are stacked into one matrix such that each column represents one sample. The goal is to find a transformation of $\mathbf{X}$ to $\mathbf{Z}$ by projecting the samples in $\mathbf{X}$ onto a hyperplane with dimension $L - 1$. Of all the possible lines the desired line is the one that maximises the separability of $\mathbf{Z}$. Finding a good projection vector requires defining a measure of separation between the projection. A solution proposed by Fisher is to maximise a function that represents the difference between the means, normalised by a measure of the within-class variability. The resulting function is called the Fisher linear discriminant and is defined as the linear function $\mathbf{Z} = \mathbf{X}\mathbf{U}^*$ where $\mathbf{Z}$ is a matrix of the resulting $L - 1$ discriminants. The idea is to find a projection, $\mathbf{U}^*$, where examples from the same class are projected very close to each other and, at the same time, the projected means are as far apart as possible. Algorithm 2.2 describes how the linear discriminants are

obtained. Each of the linear discriminants are a linear combination of the feature variables.

Algorithm 2.2 Derivation of Linear Discriminants:

1. Collect samples of data sets for the different L classes.
2. Compute the p-dimensional mean vectors for the different L classes for the data set.
3. Compute $\mathbf{S}_{n-1_1}, \mathbf{S}_{n-1_2}, .., \mathbf{S}_{n-1_L}$, the $p \times p$ covariance matrices for the different classes.
4. Compute the between-class scatter matrix $\underset{p\times p}{\mathbf{S}_B} = \sum_{i=1}^{L} n_i \left(\bar{\mathbf{x}}_i - \bar{\mathbf{x}}\right)\left(\bar{\mathbf{x}}_i - \bar{\mathbf{x}}\right)'$ and the within-class scatter matrix $\underset{p\times p}{\mathbf{S}_W} = \sum_{i=1}^{L} (n_i - 1)\mathbf{S}_{n-1_i}$ where $\bar{\mathbf{x}}$ is the overall mean, $\bar{\mathbf{x}}_i$ and n_i are the sample mean and sizes of the respective classes.
5. Compute the eigenvalue and eigenvector pairs which satisfy the following condition: $\mathbf{A}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$ where $\mathbf{A} = \mathbf{S}_W^{-1}\mathbf{S}_B$, where λ_i demotes the i^{th} eigenvalue and $\boldsymbol{v}_i$ denotes the associated eigenvector.
6. Construct $\underset{p\times p}{\mathbf{U}}$ whose columns are the orthonormal eigenvectors, sorted by decreasing eigenvalues.
7. Choose $m < p$ and derive $\underset{p\times m}{\mathbf{U}^*}$ by taking the first m columns of $\mathbf{U}$.
8. Use $\mathbf{U}^*$ to transform the samples onto the new subspace, that is compute $\underset{n\times m}{\mathbf{Z}} = \mathbf{X}\mathbf{U}^*$.

$\mathbf{Z}$ is a reduced feature space representation of $\mathbf{X}$ that may contain most of the information contained in $\mathbf{X}$. The discriminants can be used to classify unknown objects.

Example 2.2. Consider a data set consists of five observations of two feature variables, adapted from Elhabian & Farag (2009). The observations can be divided into two classes. Samples of data sets for the two classes were collected. The samples for class one, denoted by $\mathbf{X}_1$, and class two, denoted by $\mathbf{X}_2$, are

$$\mathbf{X}_1 = \begin{bmatrix} 4 & 2 \\ 2 & 4 \\ 2 & 3 \\ 3 & 6 \\ 4 & 4 \end{bmatrix}, \mathbf{X}_2 = \begin{bmatrix} 9 & 10 \\ 6 & 8 \\ 9 & 5 \\ 8 & 7 \\ 10 & 8 \end{bmatrix}.$$

The mean vectors for the different classes of this data set are $\bar{\mathbf{x}}_1' = \begin{bmatrix} 3 & 3.8 \end{bmatrix}$ and $\bar{\mathbf{x}}_2' = \begin{bmatrix} 8.4 & 7.6 \end{bmatrix}$. The covariance matrix of the first class is $\mathbf{S}_1 = \sum_{\mathbf{x}\in\text{class } 1} \left(\mathbf{x} - \bar{\mathbf{x}}_1\right)\left(\mathbf{x} - \bar{\mathbf{x}}_1\right)' = \begin{bmatrix} 1 & -0.25 \\ -0.25 & 2.2 \end{bmatrix}$. The covariance matrix of the second class is $\mathbf{S}_2 = \sum_{\mathbf{x}\in\text{class } 2} \left(\mathbf{x} - \bar{\mathbf{x}}_2\right)\left(\mathbf{x} - \bar{\mathbf{x}}_2\right)' =$

$\begin{bmatrix} 2.3 & -0.05 \\ -0.05 & 3.3 \end{bmatrix}$. The within-class scatter matrix is given by $\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2 = \begin{bmatrix} 3.3 & -0.3 \\ -0.3 & 5.5 \end{bmatrix}$. The between-class scatter matrix is given by $\mathbf{S}_B = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)' = \begin{bmatrix} 29.16 & 20.52 \\ 20.52 & 14.44 \end{bmatrix}$. The eigenvalues are calculated by solving $\left|\mathbf{S}_W^{-1}\mathbf{S}_B - \lambda\mathbf{I}\right| = 0$. The eigenvalues are $\lambda_1 = 12.2007$ and $\lambda_2 = 0$. $\mathbf{U}$ is the matrix whose columns are the corresponding orthonormal eigenvectors sorted by decreasing eigenvalues, thus $\mathbf{U} = \begin{bmatrix} 0.9088 & -0.5755 \\ 0.4173 & 0.8178 \end{bmatrix}$. Let $m = 1 < d = 2$, then $\mathbf{U}^* = \begin{bmatrix} 0.9088 \\ 0.4173 \end{bmatrix}$. The new sample space is given by

$$\mathbf{Z}_1 = \mathbf{X}_1\mathbf{U}^* = \begin{bmatrix} 4 & 2 \\ 2 & 4 \\ 2 & 3 \\ 3 & 6 \\ 4 & 4 \end{bmatrix} \begin{bmatrix} 0.9088 \\ 0.4173 \end{bmatrix} = \begin{bmatrix} 4.4698 \\ 3.4868 \\ 3.0695 \\ 5.2302 \\ 5.3044 \end{bmatrix}$$

and

$$\mathbf{Z}_2 = \mathbf{X}_2\mathbf{U}^* = \begin{bmatrix} 9 & 10 \\ 6 & 8 \\ 9 & 5 \\ 8 & 7 \\ 10 & 8 \end{bmatrix} \begin{bmatrix} 0.9088 \\ 0.4173 \end{bmatrix} = \begin{bmatrix} 12.3522 \\ 8.7912 \\ 10.2657 \\ 10.1915 \\ 12.4264 \end{bmatrix}$$

which are 5×1 matrices, smaller than the original 5×2 matrices. The discriminant vectors, $\mathbf{Z}_1$ and $\mathbf{Z}_2$, can be used as the input or feature variables in other classification techniques, for example neural networks and support vector machines.

2.4 Summary: PCA and LDA

PCA and LDA are both linear transformation techniques that are commonly used in dimension reduction but are quite different. PCA can be described as an unsupervised algorithm. An unsupervised algorithm is an algorithm that only utilises predictor variables in the training or estimation of the model (Kun et al., 2006). PCA is an unsupervised algorithm since it ignores class labels and its goal is to find the principal components that maximise the variance in the data set. LDA is a supervised algorithm. A supervised algorithm is an algorithm that has both predictor variables and their response in the training data (Laskov et al., 2005). LDA is a supervised algorithm since it uses the class labels to compute the linear discriminants that will represent the axes that maximises the separation between the classes. LDA often assumes normally distributed data whereas PCA makes no assumptions about the data

(James et al., 2013, pg 138).

It is difficult to know which dimension reduction technique to use in a scenario that requires a dimension reduction technique. If the response $\mathbf{Y}$ is given, one is advised to use the LDA approach otherwise the PCA approach should be used (Martínez & Kak, 2001). In most cases using LDA when the response is given is best but there are instances where PCA outperforms LDA even when the response is given (Martínez & Kak, 2001).

2.5 PCA and LDA Applied to the Wine Data Set

Consider the wine data set discussed in section 2.1 on page 16. A classification tree was used as the classifier of the type of wine using the thirteen feature variables. The data set was split into a train and test set using a 70/30 proportion, that is 70% of the data set was used to train the model and the rest was reserved as the test set. A simple classification tree model was fitted on the training set. The train and test accuracy of the model is shown in table 2.2 on page 30. More detailed results of all the models developed in this chapter are available in appendix A.6 on page 110. The classifier attained a test accuracy of 82.69%.

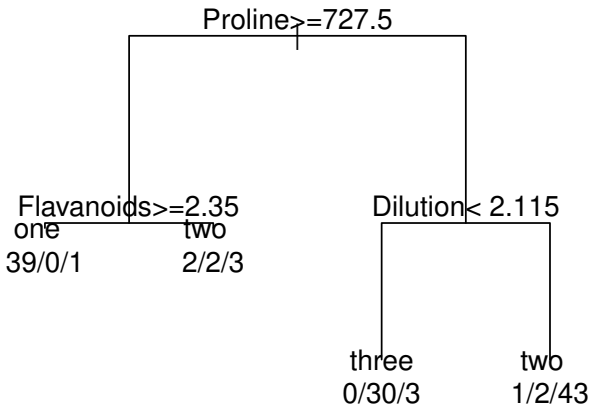


Figure 2.4: Tree plot of the classifier.

Figure 2.4 provides a plot of this classifier. The plot indicates that if the proline quantity of the wine is greater than or equal to 727.5 and the quantity of flavanoids in the wine is greater than or equal to 2.35 it is classified as belonging to class one. If the proline quantity of the wine is greater than or equal to 727.5 and the quantity of flavanoids in the wine is less than 2.35 it is classified as belonging to class two. On the other hand if the proline quantity of the wine is less than 727.5 and the quantity of D280 or OD315 in diluted wines is less than

2.115 it is classified as belonging to class three. If the proline quantity of the wine is less than 727.5 and the quantity of D280 or OD315 in diluted wines is greater than or equal to 2.115 it is classified as belonging to class two. From the plot it is clear to see that the quantity of proline and flavanoids in the wine and the quantity of D280 or OD315 in diluted wines were at the top in terms of variable importance during the training process.

The next step is to reduce the feature space of the model and assess any differences in classifier performance. The first reduction technique that will be utilised is PCA. The results of the PCA, available in table A.4 in appendix A.4 on page 104, are quite difficult to interpret. The biplot of the principal components provided in figure 2.5 is also difficult to interpret. The table indicates that the first principal component is mostly made up of flavanoids, phenols, dilution and nonflavanoids. The second principal component is mostly made up of colour, alcohol, proline and ash. However it is very difficult to assess the results unless a wine expert or chemist is consulted.

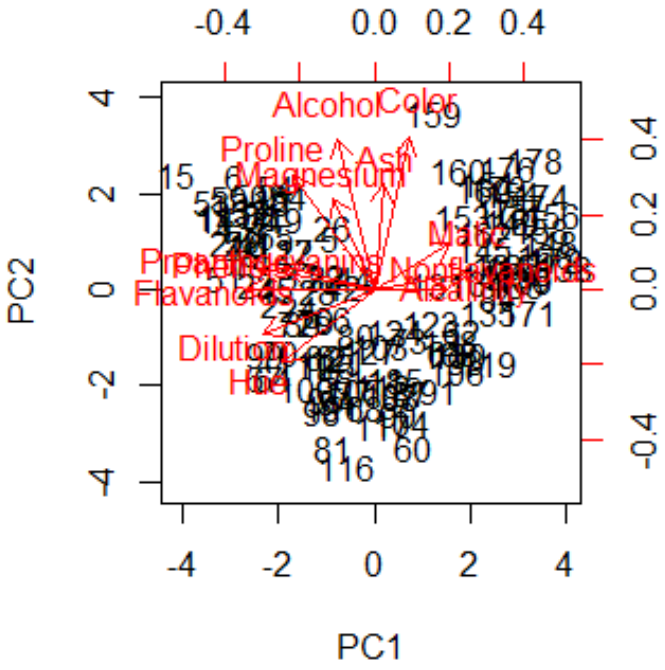
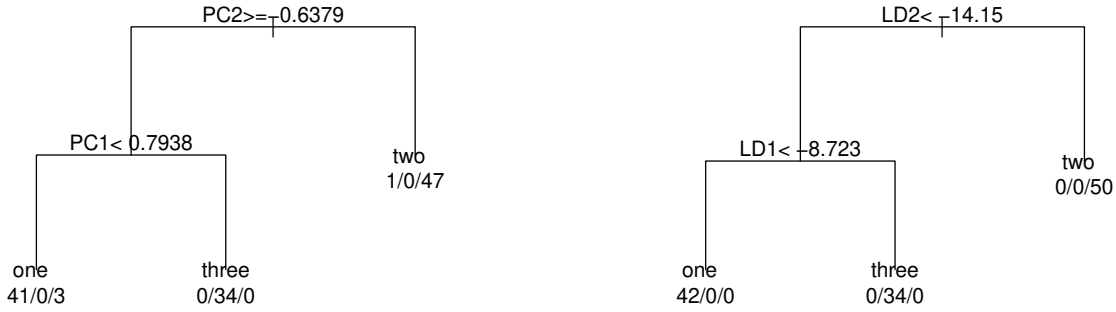


Figure 2.5: The biplot of the PCA for the wine data set.

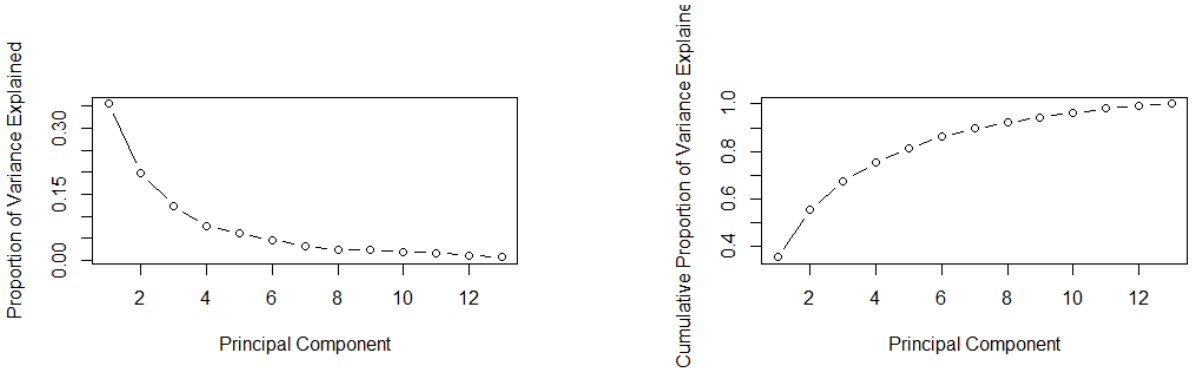


(a) Tree plot of the classifier on reduced feature space using PCA.

(b) Tree plot of the classifier on reduced feature space using LDA.

Figure 2.6: Tree plot of the classifier on reduced feature space.

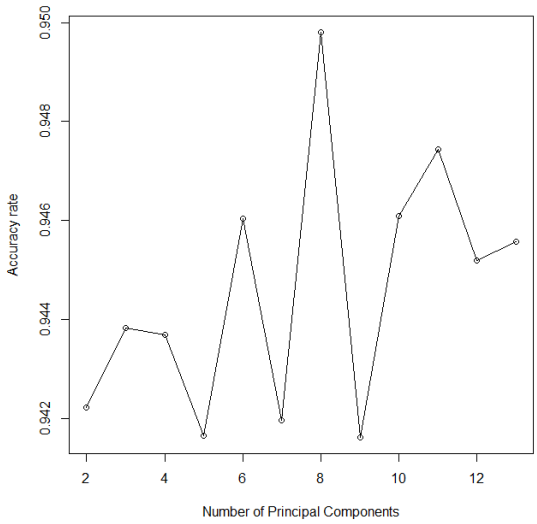
A classification tree model was fit on the reduced feature space. The number of principal components used is eight. The scree plot and cumulative plot (figures 2.7a and 2.7b) were used to determine the number of principal components that should be utilised in the classifier. Figure 2.7a shows that as you approach eight principal components the graph starts to converge and because of this eight principal components were used to fit the model. In addition an investigation was conducted to determine the number of principal components that result in the best model. This analysis confirmed that using eight principal components to train the classifier resulted in the highest accuracy (see figure 2.8a). However the accuracy rates were quite similar. The R code for the investigation is available in appendix A.7 on page 121.



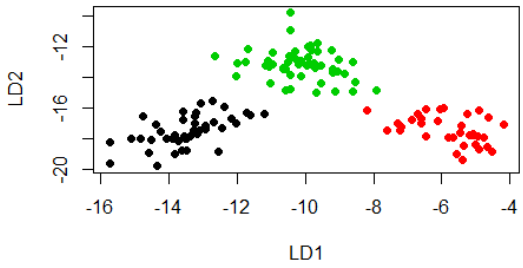
(a) Scree plot.

(b) Cumulative scree plot.

Figure 2.7: Principal components analysis results.



(a) Classification tree accuracy for various numbers of principal components.



(b) Projection of the reduced feature space.

Figure 2.8: Wine data accuracy and reduced feature space.

The training and test accuracy of the optimal model is given in table 2.2 and shows that applying PCA to the data set resulted in an improved test accuracy of 92.31%. This is definitely an improvement and it demonstrates the power of PCA as a dimension reduction technique and how it can result in improved model performance. Figure 2.6a provides a plot of this model. The plot indicates that if the second principal component of the observation is less than -0.6379 then the observation is classified as belonging to class two. If the second principal component of the observation is greater than or equal to -0.6379 and the first principal component of the observation is less than 0.7938 the observation is classified as belonging to class one. If the second principal component of the observation is greater than or equal to -0.6379 and the first principal component of the observation is greater than or equal to 0.7938 the observation is classified as belonging to class three. The first and second principal components were definitely important variables in this model. The full classification tree results are available in appendix A.6 on page 110. After reducing the data set using LDA a classification tree model was built using the reduced feature space. Figure 2.8b shows how the classes are distributed on the LDA reduced feature space made up of two linear discriminants. Clearly LDA produces a feature space which separates the classes.

The results of the classifier using the LDA reduced feature space are given in table 2.2. The model had a test accuracy of 98.08% which is the highest test accuracy for the three tree classifiers. Using LDA to reduce the feature space of this data set improves model performance. The results in table 2.2 show that LDA is the better reduction technique in the context of this example and reduction techniques considered. As discussed in section 2.4 this could be due to the fact that classification and LDA as a reduction method are both supervised techniques whereas PCA is an unsupervised technique. Figure 2.6b provides a

plot of the tree classifier constructed using the linear discriminants. If the second linear discriminant of the observation is greater than or equal to -14.15 then it is classified as belonging to class two. If the second linear discriminant of the observation is less than -14.15 and the first linear discriminant of the observation is less than -8.723 the observation is classified as belonging to class one. If the second linear discriminant of the observation is less than -14.15 and the first linear discriminant of the observation is greater than or equal to -8.723 the observation is classified as belonging to class three. The full classification tree results are available in appendix A.6 on page 110.

Table 2.2: Classification tree results.

Model	Train accuracy	Test accuracy
Classification tree	0.9127	0.8269
Classification tree PCA	0.9683	0.9231
Classification tree LDA	1.0000	0.9808

Repeated 7-fold cross validation was used to ensure reliability of these results, the results are provided in table 2.3. All three models had a slightly higher accuracy using repeated 7-fold cross validation. The dimension reduction techniques were still effective in improving model performance with LDA still being the best performing technique.

Table 2.3: 7-fold cross validation accuracy for the wine classification trees.

Model	Accuracy
Classification tree	0.8600
Classification tree PCA	0.9509
Classification tree LDA	0.9944

Chapter 3

Neural Networks

Machines are used to carry out many tasks that used to be performed by humans and have significantly reduced the average person's amount of hard labour, for example people used to hand wash clothes but since the invention of washing machines most people have completely stopped hand washing their clothes. Machines are really fast at completing tasks but when they were introduced they could not complete complicated tasks such as pattern recognition in real time (Fausett, 1994, pg1). On the other hand the human brain can recognise familiar patterns almost instantaneously (Veelenturf, 1995, pg 1). This is because the human brain parallel processes information. Cognitive science focuses on the study of interpretation of thought processes resulting from exposure to some type of input and learning which is the accumulation of knowledge derived from studying examples (Izenman, 2008, pg 315). The brain is arguably the most complex organ in the human body which makes the task of understanding it difficult (Müller et al., 1995, pg 12). Ramón y Cajál (1911) (as cited in Haykin (1994, pg 1)) pioneered the research on understating the human brain and introduced the idea of neurons as the structural constituents of the brain. Neurons, or nerve cells, form the building blocks of the nervous system (Izenman, 2008, pg 316). Neurons are typically five to six orders slower than silicon logic, the building blocks of machines (Haykin, 1994, pg 1). The brain has a large number of neurons with numerous interconnections between them which makes up for the slow operation rate of a neuron compared to a silicon logic gate (Haykin, 1994, pg 1). It is estimated that there are one hundred billion neurons in the human brain with each neuron having ten thousand connections to other neurons (Hanif, 2015). This complex system makes the brain a tremendously efficient system. The brain can be likened to a computer that is highly complex, nonlinear, parallel processing and has the ability to organise neurons to perform tasks such as pattern recognition many times faster than the fastest digital computer to date (Haykin, 1994, pg 1).

The brain owes much of its efficiency to experience. The human brain has an accelerated development within the first two years after birth. During the early stages of development about one million synapses are formed every second (Haykin, 1994, pg 2). Synapses are joints which mediate the interactions between neurons. Before the 1990s man-made machines used

to perform their computations sequentially, step by step (Veelenturf, 1995, pg 1). Man-made machines are made up of many silicon logic gates or building blocks that have different and specific functions and come together to carry out the overall function of the machine. Unfortunately when one of those building blocks fails the entire machine stops working until the relevant building block is repaired. Neurons, the building blocks of the human brain, have identical functions and as a result if any one of the neurons fail the system will continue to carry out its tasks since another neuron can be trained to take over the responsibility of the damaged neuron (Fausett, 1994, pg 7). This characteristic of the brain was used as a criteria in the 1940s when researching the possibility of designing machines that would put the world one step closer to having more intelligent systems (Müller et al., 1995, pg 13). The introduction of artificial neural networks, commonly referred to as neural networks, was a first step in this direction.

3.1 Human Nervous System

The human nervous system can be divided into two main parts: The peripheral nervous system (PNS) and the central nervous system (CNS). The PNS carries information to and from the CNS. The PNS consists of sensory and motor neurons. Sensory neurons are nerve cells responsible for converting external stimuli from the organism's environment into information that is transported to the CNS via receptors. Motor neurons are the nerve cells responsible for carrying messages from the CNS out to the muscles and glands via effectors. The CNS consists of the brain and the spinal cord. Haykin (1994, pg 6) demonstrates that the human nervous system can be split into a three-stage system as shown in figure 3.1 (adapted from Haykin (1994, pg 6)). Central to the system is the brain or cerebral cortex represented by the neural network block. The cerebral cortex consists of a vast network of interconnected neurons (Izenman, 2008, pg 316) and receives information, processes it and makes suitable decisions. In figure 3.1 the arrows going from left to right represent the incoming transmission of information bearing signals through the system. The arrows going from right to left indicate the presence of feedback in the system. The receptors convert stimuli from the human body or the external environment into electrical impulses that convey information to the neural network. The effectors convert the electrical impulses from the neural network into identifiable responses as system outputs. An example of a stimuli is touching a hot stove. Sensory nerves in the skin travel to the receptors where they are converted into a message that is transported to the brain. The brain translates the received information as pain and decides how the body should respond, for example pulling your hand away and that information is sent via the effectors. The response is sent by the motor cells back to the hand which results in someone pulling their hand away.

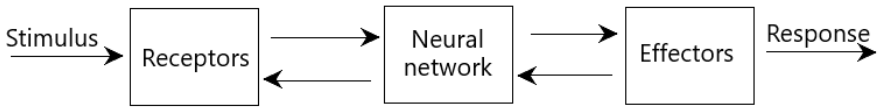


Figure 3.1: The human nervous system.

3.2 Biological Neurons

Understanding the structure of a biological neuron is fundamental in order to understand the structure of an artificial neuron. Detailed investigations of neural cells, conducted using the electron microscope, revealed that all neurons are constructed from the same basic parts independent of their size or shape (Müller et al., 1995, pg 3). A biological neuron has four key components that are of particular interest in understanding an artificial neuron, namely its dendrites, cell body or soma, axon and synapses. The dendrites are a tree like structure (see figure 3.2) which receives information or signals from other adjacent neurons and carries the signals towards the soma. The signals are electric impulses that are transmitted across a synaptic gap by means of a chemical process (Müller et al., 1995, pg 5). The chemical process adjusts the incoming signals, typically by scaling the frequency of the signals received, in a way similar to the purpose of the weights in an artificial neural network (Fausett, 1994, pg 5) as discussed in section 3.4 on page 35. The soma is the heart of the cell, contains the nucleus and is responsible for the growth and maintenance of the neuron (Hanif, 2015). The soma combines the incoming signals, similar to the combining method in an artificial neuron. When sufficient input is received from the signals, the generated neural activity is carried by the axon, a single long fibre, to other neurons across a synaptic gap referred to as a synapse (Hanif, 2015). Synapses are joints between neurons which transmit signals from one neuron to another.

It is often assumed that a cell either shoots a signal or does not at any particular point in time in order to treat transmitted signals as binary (Fausett, 1994, pg 5). This is why binary or sigmoid functions are the most common activation functions in artificial neural networks. These activation functions are also beneficial when used in neural networks due to the simple relationship between the value of the function at a point and the value of the derivative at that point which reduces the computational burden during training (Fausett, 1994, pg 17). The frequency of shooting varies and can be viewed as a signal of either greater or lesser

magnitude which corresponds to looking at discrete time steps and summing all activity at a particular point in time in an artificial neuron.

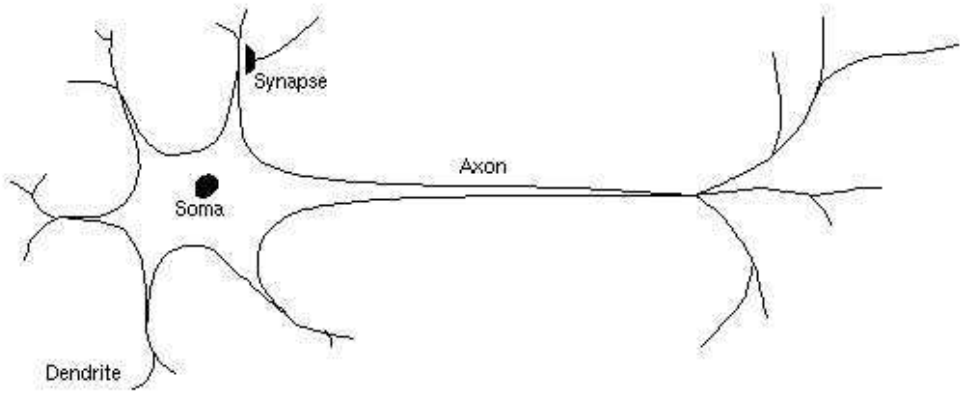


Figure 3.2: A biological neuron.

3.3 Artificial Neurons

An artificial neuron, first introduced by McCulloch & Pitts (1943), is an information-processing unit that is fundamental to the operation of an artificial neural network (Hanif, 2015). Haykin (1994, pg 8) indicates that there are three basic components of an artificial neuron:

1. Connecting links, each of which is characterised by a weight of its own;

2. A method, typically adding, of combining input signals which are weighted by the respective connecting links of the artificial neuron;

3. An activation function for limiting the amplitude of the output of the neuron.

Consider the basic structure of an artificial neuron as shown in figure 3.3. Like the synapses of a biological neuron, the connecting links provide a pathway for input signals coming from an input source or surrounding neurons to enter the neuron. The input signals are adjusted by the weight of the connecting link they are traveling through. The adjusted input signals travel to the accumulator. Similar to the function of the soma in the biological neuron, the accumulator uses a method to combine the incoming input signals and the result of this is referred to as the net input of the neuron. The net input travels to the activation function where the activation function is applied to the net input and the resulting value is the output of the neuron which is often referred to as the activation of the neuron.

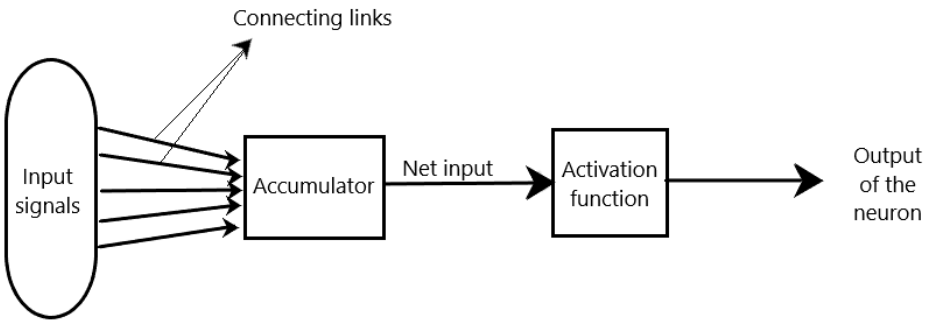


Figure 3.3: The basic structure of an artificial neuron.

A neuron sends its activation as an input signal to several other surrounding neurons. Only one activation signal is sent at a time. Consider figure 3.4 (adapted from Fausett (1994, pg 4)). Neuron Ne_4 receives input signals from neurons Ne_1, Ne_2 and Ne_3 . The activations or output signals of these neurons are denoted X_1, X_2, and X_3 respectively. The weights on the connections from Ne_1, Ne_2 and Ne_3 to Ne_4 are w_1, w_2 and w_3 respectively. The net input heading towards neuron Ne_4 is the sum of the weighted signals from neurons Ne_1, Ne_2 and Ne_3, namely $w_1X_1 + w_2X_2 + w_3X_3$. The output, Y, released by Ne_4 is the result of passing the net input through the activation function of neuron Ne_4.

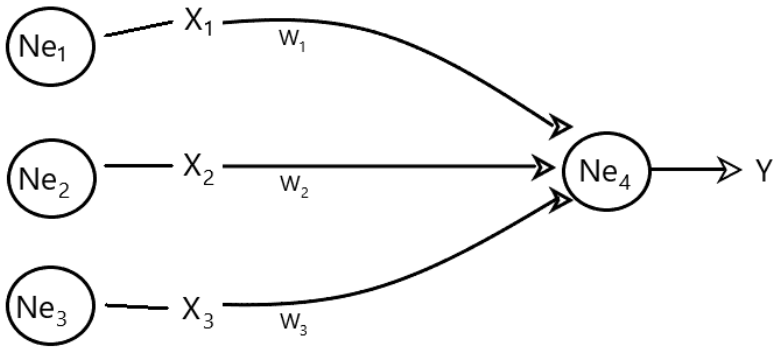


Figure 3.4: Artificial neurons (adapted from Fausett, 1994, pg 4).

3.4 Artificial Neural Networks

Artificial neural network models, or neural networks (NN), are algorithms for cognitive tasks which are loosely based on concepts derived from research into the nature of the brain (Müller et al., 1995, pg 13). In general, a neural network is a machine designed to mimic how the brain carries out certain functions. Müller et al. (1995, pg 13) defines a neural network as a directed graph with the following properties:

1. A state variable, the net input of the i^{th} neuron N_i, associated with each node i;

2. A real-valued weight, w_{ik}, associated with each connecting link (ik) between two nodes i and k;

3. A real-valued bias, b_i, associated with each node i;

4. For each node, i, a transfer or activation function, $f_i\,[N_k,\, w_{ik},\, b_i,\, (k \neq i)]$, is defined to determine the state of the node as a function of its bias, the weights of its incoming links and of the states of the nodes connected to it by these links.

The nodes refer to the neurons, the links refer to the synapses and the bias is known as the activation threshold (Dongare et al., 2012). Nodes without links towards them are called input neurons, for example the nodes Ne_1, Ne_2, Ne_3 in figure 3.4, and nodes without links away from them are called output neurons, for example node Ne_4 in figure 3.4. Some nodes can be input and output neurons.

Suppose there exists an input vector $\mathbf{X} = \begin{bmatrix} X_1 & X_2 & ... & X_n \end{bmatrix}'$ where $X_i \in \mathbb{R}$, transported through connecting links, or synapses, to an accumulator (figure 3.5). The structure of a neuron, from the input signal received from the connecting links to the output of the neuron Y is shown in figure 3.5. Each connecting link, k, has a synaptic weight w_k attached to it. The input signal X_k traveling through connecting link k is multiplied by the synaptic weight w_k which results in $w_k X_k$ being the signal reaching the accumulator from the k^{th} input connection. All the signals are summed up in the accumulator and yield $w_1X_1 + w_2X_2 + ... + w_nX_n = \sum_{k=1}^{n} w_kX_k$. A bias is also transported to the accumulator represented by an additional weight $b = w_0$ applied to input signal $X_0 = 1$. The net input of the neuron, denoted by N, is

$$\begin{aligned} N =& b(1) + w_1X_1 + w_2X_2 + ... + w_nX_n \\ =& w_0X_0 + w_1X_1 + w_2X_2 + ... + w_nX_n \\ =& \sum_{k=0}^{n} w_kX_k = \mathbf{W}'\mathbf{X} \end{aligned}$$

where $\mathbf{W} = \begin{bmatrix} b & w_1 & w_2 & ... & w_n \end{bmatrix}'$ is the weight vector and the input vector is adjusted to include the bias term as $\mathbf{X} = \begin{bmatrix} 1 & X_1 & X_2 & ... & X_n \end{bmatrix}'$. The net input of neuron N is transported to the activation function f which helps in finding the mapping of the neuron denoted as $\Phi\,(N)$. The activation function maps the value of N to some finite interval such as $[0, 1]$ or $[-1, 1]$ which is referred to as the output of the neuron. The value Y denotes the output of the neuron and is referred to as the activation of the neuron. Figure 3.6 illustrates an example of the activation of an activation function used by a neuron where $f\,(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$. In this instance the activation function f is called a hard limit function (hardlim) and transforms the value of N to the finite interval $[0, 1]$.

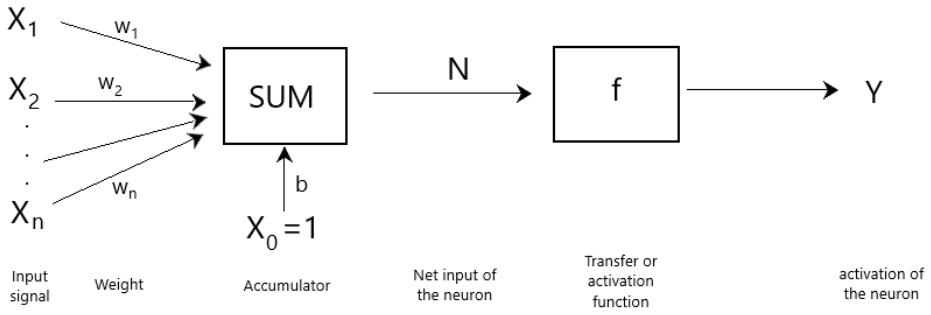


Figure 3.5: General structure of a neuron (adapted from Jäger, 2005).

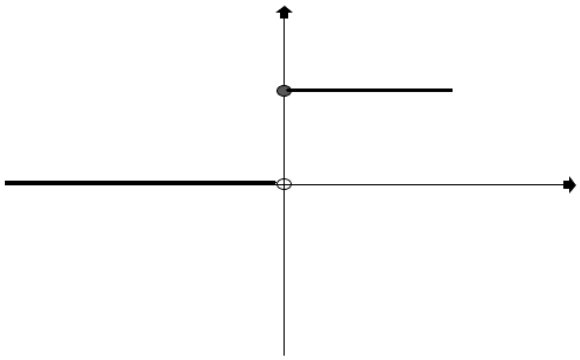


Figure 3.6: The hard limit activation function.

Example 3.1. Consider the neuron shown in figure 3.7. In this neuron $f(x) = 1$ if $x \geq 0$ or $f(x) = 0$ if $x < 0$, $w_1 = -1$, $w_2 = 1$, $b = -\frac{1}{2}$, $X_1 = \frac{1}{4}$ and $X_2 = \frac{3}{4}$. The mapping of the neuron is

$$\Phi(N) = \begin{cases} 1, & w_0X_0 + w_1X_1 + w_2X_2 \geq 0 \\ 0, & w_0X_0 + w_1X_1 + w_2X_2 < 0. \end{cases}$$

The net input of the neuron N is calculated as follows

$$\begin{aligned} N = \sum_{k=0}^{n} w_kX_k &= w_0X_0 + w_1X_1 + w_2X_2 \\ &= -\frac{1}{2}.1 - 1.\frac{1}{4} + 1.\frac{3}{4} = 0. \end{aligned}$$

The activation of the neuron y is given by $\Phi(N) = \Phi(0) = 1$.

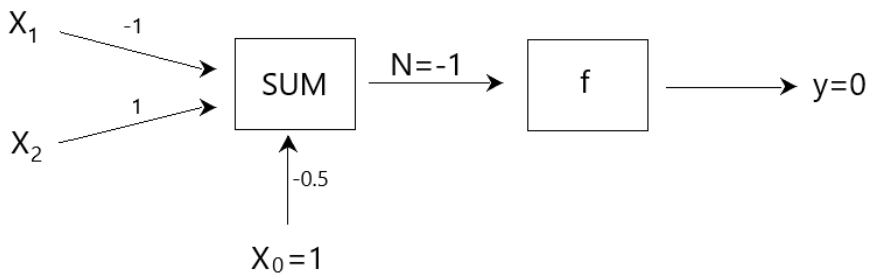


Figure 3.7: The neuron in example 1.1.

In general there are more than one neurons in a NN. Consider figure 3.8, a NN with two neurons, N_1 and N_2, and two inputs, X_1 and X_2. The weights of the connecting links between neuron one and the inputs are w_{11} and w_{12}. The weights of the connecting links between neuron two and the inputs are w_{21} and w_{22}. For every weight w_{ij}, i represents the number of the neuron and j represents the input to that neuron. Thus w_{ij} is the weight of the connection from input j to neuron i. The net inputs of these neurons are

$$\begin{aligned} N_1 =& w_{11}X_1 + w_{12}X_2 + b_1 \\ N_2 =& w_{21}X_1 + w_{22}X_2 + b_2. \end{aligned}$$

The net inputs can be written in matrix form as

$$\mathbf{N} = \begin{bmatrix} N_1 \\ N_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_1 \end{bmatrix}.$$

$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$ is referred to as the weight matrix and $\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$ is referred to as the bias vector. An activation function is applied to the net inputs.

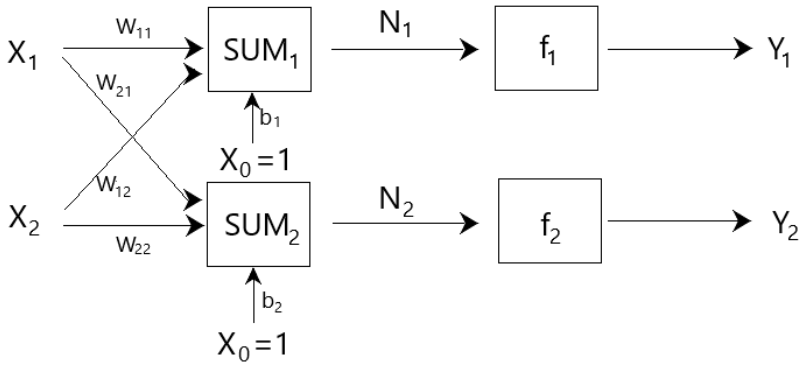


Figure 3.8: An example of a NN with two neurons.

Example 3.2. Suppose there exists a neural network with two inputs and two neurons where the weight matrix is given by $\begin{bmatrix} 1 & \frac{1}{2} \\ 2 & -1 \end{bmatrix}$ and the bias vector is given by $\begin{bmatrix} -\frac{1}{2} \\ 2 \end{bmatrix}$. If the inputs are $\mathbf{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ the net inputs, $\mathbf{N} = \begin{bmatrix} N_1 & N_2 \end{bmatrix}'$, are

$$\begin{aligned} \mathbf{N} &= \mathbf{WX} + \mathbf{b} \\ \begin{bmatrix} N_1 \\ N_2 \end{bmatrix} &= \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{2} \\ 2 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 2 \end{bmatrix} + \begin{bmatrix} -\frac{1}{2} \\ 2 \end{bmatrix} \\ &= \begin{bmatrix} -\frac{1}{2} \\ -2 \end{bmatrix}. \end{aligned}$$

The activation function is applied component wise to $\mathbf{N} = \begin{bmatrix} N_1 & N_2 \end{bmatrix}'$ and yields output $\mathbf{Y}$:

$$\mathbf{Y} = \Phi(\mathbf{N}) = \Phi\left(\begin{bmatrix} -\frac{1}{2} \\ -2 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

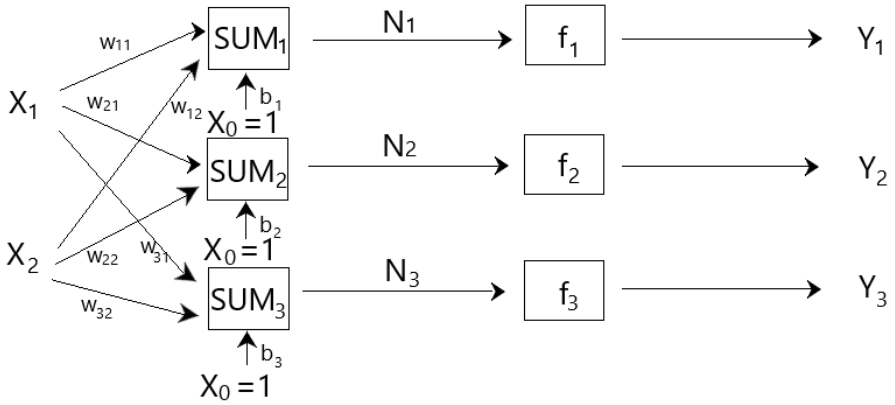


Figure 3.9: An example of a NN with three neurons.

Example 3.3. Consider the neural network with two inputs, X_1 and X_2, and three neurons, N_1, N_2 and N_3, as shown in figure 3.9. In matrix form the net inputs, $\mathbf{N} = \mathbf{WX} + \mathbf{b}$, of the neurons are

$$\begin{bmatrix} N_1 \\ N_2 \\ N_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

The weight matrix of the neuron is $\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$, a 3×2 matrix, and the bias vector is $\mathbf{b} = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}'$.

In general a neural network with n inputs, $\mathbf{X} = \begin{bmatrix} X_1 & X_2 & ... & X_n \end{bmatrix}'$, and m neurons has weight matrix

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & ... & w_{1n} \\ w_{21} & w_{22} & ... & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & ... & w_{mn} \end{bmatrix}$$

where w_{ij} represents the input to neuron i from input j and bias vector $\mathbf{b} = \begin{bmatrix} b_1 & b_2 & ... & b_m \end{bmatrix}'$. The net input is given by

$$\mathbf{N} = \mathbf{WX} + \mathbf{b}$$

$$\begin{bmatrix} N_1 \\ N_2 \\ \vdots \\ N_m \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & ... & w_{1n} \\ w_{21} & w_{22} & ... & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & ... & w_{mn} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$= \begin{bmatrix} w_{01} & w_{02} & ... & w_{0n} \\ w_{21} & w_{22} & ... & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & ... & w_{mn} \end{bmatrix} \begin{bmatrix} 1 \\ X_1 \\ \vdots \\ X_n \end{bmatrix}$$

where $X_0 = 1$ and $w_{0k} = b_k$. In general, the bias of each node, b_i, is included in the first column of the weight matrix.

3.5 Multi-layered Artificial Neural Networks

Outputs of a neural network can be used as inputs for another neural network. In this section, a simplified version of the structure of an artificial neuron as shown in figure 3.10 will be used. In figure 3.10, the circle encompasses three stages in a neural network: the accumulator, the net input of the neuron and the application of the activation function on the net input of the neuron.

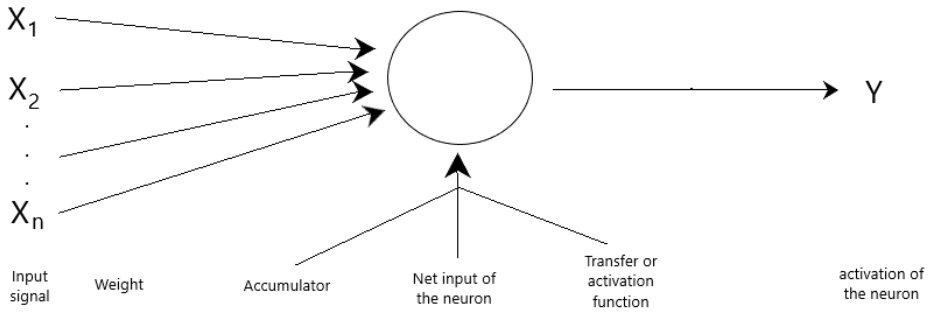
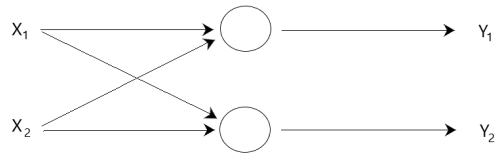
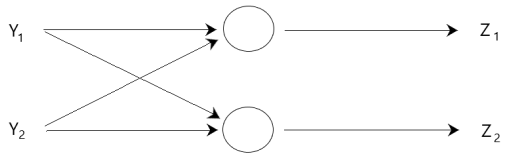


Figure 3.10: Simplified structure of an artificial neuron.

Consider the neural network represented in figure 3.8. Figure 3.11a shows the simplified structure of this neural network. The output vector, $\mathbf{Y} = \begin{bmatrix} Y_1 & Y_2 \end{bmatrix}$, of this neural network could be used as the inputs for another neural network which yields output vector $\mathbf{Z} = \begin{bmatrix} Z_1 & Z_2 \end{bmatrix}$ as shown in figure 3.11b.



(a) Simplified structure of an artificial neural network.



(b) Simplified structure of an artificial neural network.

Figure 3.11: Simplified structures of artificial neural networks.

Combining the system from the inputs, $\mathbf{X}$, in the neural network shown in figure 3.11a to the output, $\mathbf{Z}$, in the neural network shown in figure 3.11b results in a two layered neural network shown in figure 3.12a. This neural network has two weight matrices $\mathbf{W}^1$, for the first layer represented in figure 3.11a, and $\mathbf{W}^2$, for the second layers represented in figure 3.11b. The neural network has two bias vectors, $\mathbf{b}^1$ and $\mathbf{b}^2$, for the first and second layer of the neural network respectively which are included in the weight matrices. The output of the first layer is given by

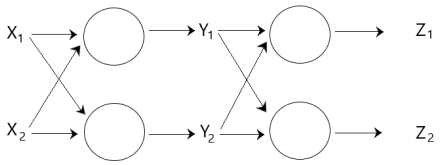
$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = f^1 \left(\mathbf{W}^1 \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \right)$$

where f^1 is the activation function of the first layer. The output of the second layer, the final

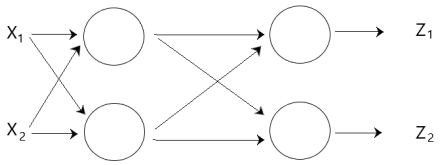
output of the two layered system, is given by

$$\mathbf{Z} = \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = f^2 \left(\mathbf{W}^2 \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \right)$$

where f^2 is the activation function of the second layer. The outputs in the middle of the multi layered neural network, that is $\mathbf{Y}$ in figure 3.12a, are referred to as the hidden layer and are usually not labeled in the diagram as is shown in figure 3.12b.



(a) A two-layered neural network.



(b) A two-layered neural network.

Figure 3.12: Examples of two-layered neural networks.

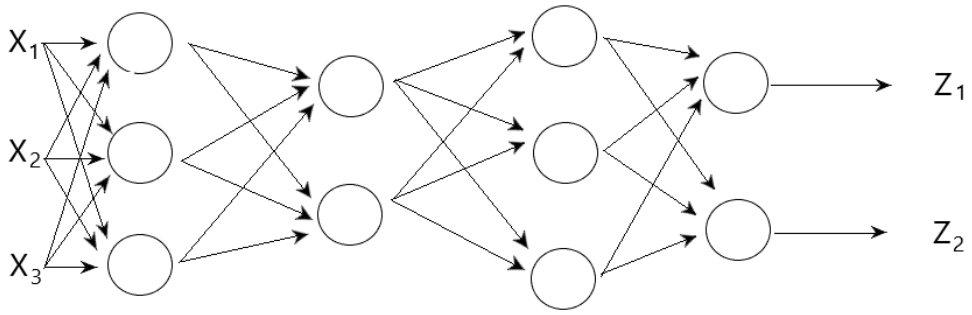


Figure 3.13: An example of a four-layered neural network.

Figure 3.13 illustrates a multi-layered neural network with two hidden layers. The input vector, $\mathbf{X}$, is mapped to an output vector, $\mathbf{Z}$. A neural network can be conceptualised as a mapping between a $n-$dimensional space to a real valued vector. The input vector for the NN shown in figure 3.13, can be thought of as objects whose features are extracted, measured and stored in the feature vector $\mathbf{X} \in \mathbb{R}^3$, as numerical values in this instance, which are classified into an output vector $\mathbf{Z} \in \mathbb{R}^2$ that may be in the form of classes, in this instance two classes. If the feature vector is an element of the vector space $\mathbb{R}^n$ and the output vector is an element of vector space $\mathbb{R}^l$ then the following lemma holds (Veelenturf, 1995, pg 10).

Lemma 3.4. *A neural network with n inputs and l outputs is nothing else than a computing machine for a mapping*

$$\Phi : \mathbb{R}^n \to \mathbb{R}^l$$

This mapping depends on the weights and biases of the neural network.

Thus neural networks can be used to perform classifications. This can be done by collecting samples of feature vectors for each class, termed the training set, and training the neural network by adapting the parameters, namely the weights and biases, of the neural network in a way that results in the correct classification of the objects. Neural networks can be used as a supervised learning model. When a new observation is collected its features are extracted, measured and stored into a feature vector of the object. The feature vector is presented to the neural network which classifies the object into the predicted class.

3.6 Perceptrons

A perceptron is a neural network that has one layer of neurons and the hardlim function as the activation function. Example 3.2 on page 38 provides a simple illustration of a perceptron with weight vector $\mathbf{W}' = \begin{bmatrix} -\frac{1}{2} & -1 & 1 \end{bmatrix}$ which includes bias $b = -\frac{1}{2}$. In example 3.2, the mapping of the neural network is

$$\Phi(\mathbf{X}) = \begin{cases} 1, & -\frac{1}{2} - X_1 + X_2 \geq 0 \\ 0 & -\frac{1}{2} - X_1 + X_2 < 0. \end{cases}$$

If observations with feature variables are such that $\Phi(\mathbf{X}) = 0$ then this obervation is assigned to class zero. Observations with feature variables such that $\Phi(\mathbf{X}) = 1$ are assigned to class one, resulting in a binary classifier.

Suppose features were extracted from ten observations which resulted in a two-dimensional data set. Each observation belongs to one of the two classes. Table 3.1 shows how the perceptron in example 3.2 maps the given data set where $\mathbf{Y}$ is the net output of the neural network.

Table 3.1: Input-output mapping of a perceptron.

Y	0	1	0	0	1	1	0	0	0	1
X_1	0	0	1	1	1	0	2	2	2	2
X_2	0	1	0	1	2	3	0	1	2	3

Figure 3.14 represents the mapping graphically. Points assigned to class zero are labeled with a circle and points assigned to class one are labeled with a square. The points can be separated into two different classes by a decision boundary which is the straight line $-\frac{1}{2} - X_1 + X_2 = 0$. This line forms the decision boundary in that points are classified as belonging to class zero or class one depending on which side of the line they fall. The decision boundary has normal vector $\begin{bmatrix} -1 & 1 \end{bmatrix}'$ which leads to the conclusion that the weight matrix

of a perceptron is the normal vector of the decision boundary and the bias determines the distance between the decision boundary and the origin (Freund & Schapire, 1999).

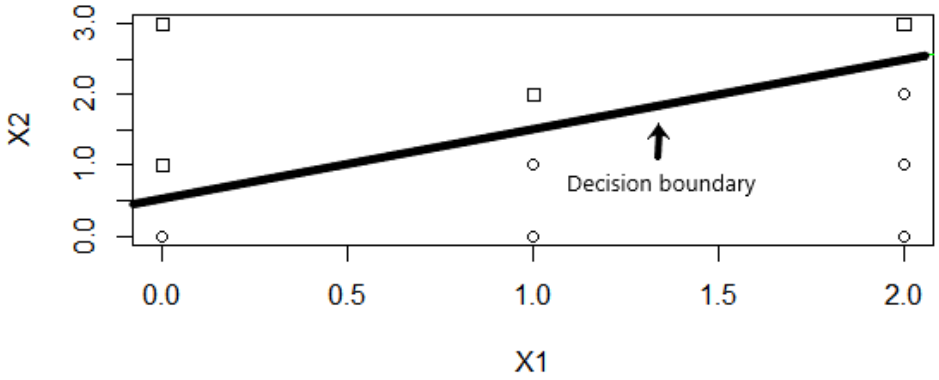


Figure 3.14: Classification by a decision boundary.

It is possible to get another decision boundary that correctly separates these observations into the two classes by adjusting the weights and bias accordingly. It is important to note that the decision boundary is not unique. The perceptron can be generalised to provide a solution for an n-dimensional data set, see figure 3.5 on page 37, assuming f is the hardlim function. The mapping of the neural network is then given by

$$\Phi(\mathbf{X}) = \begin{cases} 1, & b + w_1X_1 + w_2X_2 + ... + w_nX_n \geq 0 \\ 0 & b + w_1X_1 + w_2X_2 + ... + w_nX_n < 0 \end{cases}$$

which results in the decision boundary $b + w_1X_1 + w_2X_2 + ... + w_nX_n = 0$, a hyperplane in $\mathbb{R}^n$ with normal vector equal to weight vector $\mathbf{W}' = [w_1, w_2, ..., w_n]$. The perceptron can only correctly separate the observations into the classes if the feature vectors are linearly separable. If the the observations are linearly separable but classified incorrectly the perceptron follows a learning procedure that involves adjusting the weights and bias of the perceptron with the goal of correcting the classification of the misclassified observations. Given a learning task, the neural network can learn its parameters until it reaches a state where all observations are classified correctly. A learning task is a sample of inputs and desired outputs. Consider the learning task represented in table 3.2 with three feature samples and t the target output value. Suppose there exists a perceptron with one input and its decision boundary is given by $w_0X_0+w_1X_1 = 0$. If the observations in the sample are linearly separable a learning procedure can be used to train the perceptron in order for it to correctly separate the observations into the classes.

Table 3.2: Learning task for a perceptron with one input.

X_0	X_1	t
1	-1	0
1	0	0
1	$-\frac{1}{2}$	1

Suppose the perceptron has weight vector $\mathbf{W}' = \begin{bmatrix} \frac{1}{3} & 1 \end{bmatrix}$ which includes bias $b = \frac{1}{3}$. The mapping of the perceptron is

$$\Phi_1(\mathbf{X}) = hardlim(\frac{1}{3}X_0 + X_1).$$

The perceptron starts the first round of learning by producing outputs for the given samples. For the first sample in the data set, $\Phi_1(\mathbf{X}) = \Phi_1(1, -1) = hardlim(-\frac{2}{3}) = 0$ which is correct. For the second sample in the data set, $\Phi_1(\mathbf{X}) = \Phi_1(1, 0) = hardlim(\frac{1}{3}) = 1$ which is incorrect since $t = 0$ is the target value. This sample lies above the decision boundary when it should be below. One way of solving this problem is to change the parameters of the perceptron such that the decision boundary shifts in a counter clockwise direction, in other words, moving the normal vector away from the sample vector. This can be achieved by subtracting the sample vector from the normal vector which yields $\mathbf{W}^* = \mathbf{W} - \mathbf{X}$ where $\mathbf{W}^* = \begin{bmatrix} -\frac{2}{3} & 1 \end{bmatrix}'$ is the new weight vector. The mapping of the updated perceptron is

$$\Phi_2(\mathbf{X}) = hardlim(-\frac{2}{3}X_0 + X_1).$$

The perceptron starts the second round of learning by producing outputs for the given samples. For the first sample in the data set, $\Phi_2(\mathbf{X}) = \Phi_1(1, -1) = 0$, which is correct. The updated output of the second sample, $\Phi_2(\mathbf{X}) = \Phi_2(1, 0) = hardlim(-\frac{2}{3}) = 0$, is now correct. For the third sample in the data set, $\Phi_2(\mathbf{X}) = \Phi_2(1, \frac{1}{2}) = hardlim(-\frac{1}{6}) = 0$ which is incorrect. This can be corrected by shifting the decision boundary in a clockwise direction, that is adding the sample vector to the normal vector which yields $\mathbf{W}^{**} = \mathbf{W}^* + \mathbf{X}$ where $\mathbf{W}^{**} = \begin{bmatrix} \frac{1}{3} & \frac{3}{2} \end{bmatrix}'$, the new weight vector. The mapping of the updated perceptron is

$$\Phi_3(\mathbf{X}) = hardlim(\frac{1}{3}X_0 + \frac{3}{2}X_1).$$

The perceptron starts the third round of learning by producing outputs for the given samples. For the first sample in the data set, $\Phi_3(\mathbf{X}) = \Phi_1(1, -1) = 0$, which is correct but the output of the second sample in the data set, $\Phi_3(\mathbf{X}) = \Phi_3(1, 0) = hardlim(-\frac{1}{6}) = 0$, is now incorrect. Again the weights and bias of the perceptron are adjusted to solve the problem. This time, the sample vector is subtracted from the normal vector which yields $\mathbf{W}^{***} = \mathbf{W}^{**} - \mathbf{X}$ where $\mathbf{W}^{***} = \begin{bmatrix} -\frac{2}{3} & \frac{3}{2} \end{bmatrix}'$ the new weight vector which includes the bias. The mapping of the

updated perceptron is

$$\Phi_4(\mathbf{X}) = hardlim(-\frac{2}{3}X_0 + \frac{3}{2}X_1).$$

The perceptron starts the forth round of learning by producing outputs for the given samples. This perceptron gives the correct target value for all three sample vectors. At this point it is safe to stop and conclude that the perceptron has completed the learning procedure and responds well to all samples. $\Phi_4(\mathbf{X})$ is the final mapping and $-\frac{2}{3}X_0 + \frac{3}{2}X_1 = 0$ is the final decision boundary.

It is difficult to visualise this decision boundary when the dimension of the feature vector is more than two but it can be derived mathematically. The learning procedure can be adapted for a perceptron with more than one input. Consider the learning task presented in table 3.3 for a perceptron with n inputs. The decision boundary of the perceptron is given by $w_0X_0 + w_1X_1 + w_2X_2 + ... + w_nX_n = 0$.

Table 3.3: A learning task for a perceptron with n inputs.

X_0	X_1	X_2	$\cdots$	X_n	t
1	x_1^1	x_2^1	$\cdots$	x_n^1	t_1
1	x_1^2	x_2^2	$\cdots$	x_n^2	t_2
1	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Define an error term $e = t - a$ where $a = \Phi(\mathbf{X})$ is the calculated output of the sample. The learning procedure of the perceptron suggests that if the target value is $t = 0$ and the calculated output is $a = 1$ then $e = -1$ and hence subtract the sample vector from the weight vector which yields $\mathbf{W}^* = \mathbf{W} - \mathbf{X}$. If the target value is $t = 1$ and the calculated output is $a = 0$ then $e = 1$ and hence add the sample vector to the weight vector which yields $\mathbf{W}^* = \mathbf{W} + \mathbf{X}$. If the target value is $t = a$ then $e = 0$ and hence do nothing, that is $\mathbf{W}^* = \mathbf{W}$. In general the new weight vector after mapping each sample is defined by $\mathbf{W}^* = \mathbf{W} + e\mathbf{X}$. This learning procedure is summarised in the following algorithm (Stephen, 1990):

Algorithm 3.1 The Learning Procedure of a Perceptron.

- For sample j calculate output a_j using feature sample $\mathbf{X}^j$;
- Determine the error $e_j = t_j - a_j$;
- Update the weight vector by $\mathbf{W}^* = \mathbf{W} + e_j\mathbf{X}^j$;
- Repeat the steps until $e_j = 0$ for all observations or samples.

This algorithm was defined and discussed for the case where there is one neuron. However it can be generalised for the m neurons case. This is done by training each neuron on its own and combining the final result. Unfortunately this algorithm will only be successful if the observations are linearly separable (Freund & Schapire, 1999).

3.7 Adaptive Linear Neuron

An adaptive linear neuron (adaline) is a single layered neural network that only has one neuron and uses the identity or purelin function as its activation function (Fausett, 1994, pg 80). Like the perceptron, the adaline can be trained to separate observations into classes given a learning task. For this perceptron $e_j \in \{-1, 0, 1\}$ since it uses the hardlim function as its activation function. The adaline factors in the distance of the feature vector to the decision boundary since it uses the identity function as the activation function. The error term, e_j, is defined as $e_j = t_j - d_j$ where d_j is the output of the mapping $\Phi(\mathbf{X})$. The weight vector is updated by the rule $\mathbf{W}^* = \mathbf{W} + \eta e_j \mathbf{X}^j$ where $\eta > 0$ is referred to as a learning coefficient. This learning procedure is called the Widrow-Hoff learning algorithm (Ungar et al., 1990). Consider the learning task shown in table 3.2 on page 37. Suppose the learning rate is given by $\eta = 0.3$. For the first sample in the data set, $d_1 = \Phi_1(\mathbf{X}^1) = \Phi_1(1, -1) = purelin(-\frac{2}{3}) = -\frac{2}{3}$ which is less than the target value. $e_1 = t_1 - d_1 = \frac{2}{3}$ and the weight vector is updated as follows:

$$\mathbf{W}^* = \mathbf{W} + \eta e_1 \mathbf{X}^1 = \begin{bmatrix} \frac{1}{3} \\ 1 \end{bmatrix} + 0.3 \cdot \frac{2}{3} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.5333 \\ 0.8 \end{bmatrix}$$

which results in a new mapping $\Phi_2(\mathbf{X}^1) = 0.5333X_0 + 0.8X_1$. $\Phi_2(\mathbf{X}^1) = \Phi_2(1, -1) = -0.2667$ which is still less than the target value but closer to it which reduces the error. For the second sample $d_2 = \Phi_2(\mathbf{X}^2) = \Phi_2(1, 0) = purelin(0.5333) = 0.5333$ which is greater than the target value and $e_2 = t_2 - d_2 = 0.5333$. The weight vector is updated by

$$\mathbf{W}^{**} = \mathbf{W}^* + \eta e_2 \mathbf{X}^2 = \begin{bmatrix} 0.5333 \\ 0.8 \end{bmatrix} + 0.3 \cdot 0.5333 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.3733 \\ 0.8 \end{bmatrix}$$

which results in a new mapping $\Phi_3(\mathbf{X}^2) = 0.3733X_0 + 0.8X_1$. $\Phi_3(\mathbf{X}^2) = \Phi_3(1, 0) = 0.3733$ which is still greater than the target value but closer to it which reduces the error. For the third sample in the data set $d_3 = \Phi_3(\mathbf{X}^3) = \Phi_3(1, \frac{1}{2}) = purelin(0.7733) = 0.7733$ which is less than the target value and $e_3 = t_3 - d_3 = 0.2267$. The weight vector is updated by

$$\mathbf{W}^{***} = \mathbf{W}^{**} + \eta e_3 \mathbf{X}^3 = \begin{bmatrix} 0.3733 \\ 0.8 \end{bmatrix} + 0.3 \cdot 0.2267 \begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 0.4413 \\ 0.8340 \end{bmatrix}$$

which results in a new mapping $\Phi_4(\mathbf{X}^3) = 0.4413X_0 + 0.8340X_1$. For the fourth sample $\Phi_4(\mathbf{X}^4) = \Phi_4(1, \frac{1}{2}) = 0.8583$ which is still less than the target value but closer to it and reduces the error.

The steps can be repeated until the procedure reaches a state where the actual outputs are as close as possible to the target values and the total error over all p training samples, as given by

$$E(b, w_1) = \sum_{j=1}^{p} E_j = \sum_{j=1}^{p} (t_j - d_j)^2 = \sum_{j=1}^{p} (t_j - (b + w_1 X_1^j))^2,$$

is minimised. This error function is referred to as the sum of squared errors and depends on the weights and bias of the adaline. The aim is to find a weight w^* and bias b^* that minimises the sum of squared errors. For p samples, the mapping of the training samples is

$$\begin{bmatrix} 1 & X^1 \\ 1 & X^2 \\ 1 & \vdots \\ 1 & X^p \end{bmatrix} \begin{bmatrix} b \\ w_1 \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_p \end{bmatrix}$$

$$\mathbf{X} \begin{bmatrix} b \\ w_1 \end{bmatrix} = \mathbf{t}.$$

Multiplying both sides by $\mathbf{X}'$ yields $\mathbf{X}'\mathbf{X} \begin{bmatrix} b \\ w_1 \end{bmatrix} = \mathbf{X}'\mathbf{t}$

$$\mathbf{X}'\mathbf{X} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ X^1 & X^2 & \cdots & X^p \end{bmatrix} \begin{bmatrix} 1 & X^1 \\ 1 & X^2 \\ 1 & \vdots \\ 1 & X^p \end{bmatrix} = \begin{bmatrix} p & \sum_{j=1}^{p} X^j \\ \sum_{j=1}^{p} X^j & \sum_{j=1}^{p} (X^j)^2 \end{bmatrix}.$$

In the case that $\mathbf{X}'\mathbf{X}$ is non singular, a solution to the system is given by

$$\begin{bmatrix} b^* \\ w_1^* \end{bmatrix} = (\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{t}$$

which yields the local minimum of error function E. For the learning task given in table 3.2, $\mathbf{X} = \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & \frac{1}{2} \end{bmatrix}$, $\mathbf{t} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ and hence

$$\mathbf{X}'\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 3 & -\frac{1}{2} \\ -\frac{1}{2} & \frac{5}{4} \end{bmatrix} \text{ and}$$

$$\mathbf{X}'\mathbf{t} = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix}.$$

The optimal w_1^* and b^*, which give the local minimum of error function E, are then given by

$$\begin{bmatrix} b^* \\ w_1^* \end{bmatrix} = (\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{t} = \begin{bmatrix} 3 & -\frac{1}{2} \\ -\frac{1}{2} & \frac{5}{4} \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 0.4286 \\ 0.5714 \end{bmatrix}.$$

The mapping of the adaline is $\Phi(\mathbf{X}) = 0.4286X_0 + 0.5714X_1$.

This solution can be adapted for p training samples with an $n-$dimensional feature vector. The goal is to minimise the error function, namely

$$E(b, w_1, w_2, \cdots, w_n) = \sum_{j=1}^{p} e_j = \sum_{j=1}^{p} (t_j - d_j)^2 = \sum_{j=1}^{p} (t_j - (b + w_1 X_1^j + w_2 X_2^j + \cdots + w_n X_n^j))^2$$

For p samples the mapping of the training samples is $\mathbf{XW} = \mathbf{t}$, that is

$$\begin{bmatrix} 1 & X_1^1 & X_2^1 & \cdots & X_n^1 \\ 1 & X_1^2 & X_2^2 & \cdots & X_n^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_1^p & X_2^p & \cdots & X_n^p \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_p \end{bmatrix}.$$

Multiplying both sides with $\mathbf{X}'$ yields $\mathbf{X}'\mathbf{XW} = \mathbf{X}'\mathbf{t}$ where

$$\mathbf{X}'\mathbf{X} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ X_1^1 & X_1^2 & \cdots & X_1^p \\ X_2^1 & X_2^2 & \cdots & X_2^p \\ \vdots & \vdots & \vdots & \vdots \\ X_n^1 & X_n^2 & \cdots & X_n^p \end{bmatrix} \begin{bmatrix} 1 & X_1^1 & X_2^1 & \cdots & X_n^1 \\ 1 & X_1^2 & X_2^2 & \cdots & X_n^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_1^p & X_2^p & \cdots & X_n^p \end{bmatrix} = \begin{bmatrix} p & \sum_{j=1}^{p} X_1^j & \cdots & \sum_{j=1}^{p} X_n^j \\ \sum_{j=1}^{p} X_1^j & \sum_{j=1}^{p} \left(X_1^j\right)^2 & \cdots & \sum_{j=1}^{p} X_1^j X_n^j \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{j=1}^{p} X_n^j & \sum_{j=1}^{p} X_1^j X_n^j & \cdots & \sum_{j=1}^{p} (X_n^j)^2 \end{bmatrix}.$$

In the case where $\mathbf{X}'\mathbf{X}$ is non singular, a solution to the system is given by $\mathbf{W}^* = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{t}$ which yields the local minimum of error function E.

Another way to minimise the error would be through the use of the method of steepest descent also called the gradient descent method (Golden, 1986). Steepest descent optimisation is an algorithm for finding the nearest local minimum of a function, assuming the gradient of the function can be computed. The method of steepest descent starts at a point x_0 and, as many times as needed, moves from point x_i to the next point x_{i+1} by minimising along the line extending from point x_i in the direction of the negative of the gradient at point x_i. Suppose there exists a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. The direction of the steepest descent is the vector $-\nabla f(\mathbf{X}^0)$. Consider the function $h(s) = f(\mathbf{X}^0 + s\mathbf{u})$, where $\mathbf{u}$ is a unit vector. By the chain rule

$$\begin{aligned} h'(s) &= \frac{\partial f}{\partial X_1}\frac{\partial X_1}{\partial s} + \cdots + \frac{\partial f}{\partial X_n}\frac{\partial X_n}{\partial s} \\ &= \frac{\partial f}{\partial X_1}u_1 + \cdots + \frac{\partial f}{\partial X_n}u_n = \nabla f\left(\mathbf{X}^0 + s\mathbf{u}\right) \cdot \mathbf{u} \end{aligned}$$

and hence $h'(0) = \nabla f(\mathbf{X}^0) \cdot \mathbf{u} = ||\nabla f(\mathbf{X}^0)|| \cos\theta$, where θ is the angle between $\nabla f(\mathbf{X}^0)$ and $\mathbf{u}$. $h'(0)$ is minimised when $\theta = \pi$ which yields $\mathbf{u} = \frac{\nabla f(\mathbf{X}^0)}{||\nabla f(\mathbf{X}^0)||}$ and $h'(0) = -||\nabla f(\mathbf{X}^0)||$.

Minimising f is the same as finding the minimum of $h'(s)$ for this choice of $\mathbf{u}$. That is choose an input vector $\mathbf{X}^0$ as a starting point. Choose a value s, $s > 0$, such that $f(\mathbf{X}^0 + s\nabla f(\mathbf{X}^0)) < f(\mathbf{X}^0)$. Update the input vector by rule $\mathbf{X}^1 = \mathbf{X}^0 + s\nabla f(\mathbf{X}^0)$ and repeat the last step. This will result in a series of input vectors $(\mathbf{X}^0, \mathbf{X}^1, \mathbf{X}^2, \ldots)$ that satisfy the condition $f(\mathbf{X}^{j+1}) < f(\mathbf{X}^j) < \ldots < f(\mathbf{X}^0)$. The method should stop after a certain number of iterations or when the function values $|f(\mathbf{X}^{j+1}) - f(\mathbf{X}^j)|$ are sufficiently small or when $||\mathbf{X}^{j+1} - \mathbf{X}^j|| < \delta$ (Golden, 1986).

3.8 Multi-layered Perceptrons and Adalines

The neural networks that have been discussed thus far are referred to as feed-foward neural networks and are restricted in their application as they can only classify linearly separable patterns. Feed-foward means the signals are only transported in one direction (Jäger, 2005). Consider table 3.4a which is a truth table that represents the XOR problem (adapted from Jäger (2005)). Suppose A and B are propositions with value 1 if the proposition is true and value 0 if the proposition is false. The XOR operation, denoted by $A \oplus B$, is true if and only if one proposition, either A or B, is true but not both. The goal is to build a neural network that represents and solves the problem. Table 3.4b represents the problem in the form of a learning task.

Table 3.4: The XOR-problem.

(a) A truth table of the XOR problem.

A	B	$A \oplus B$
1	1	0
1	0	1
0	1	1
0	0	0

(b) Learning task of the XOR-problem.

X_1	X_2	t
1	1	0
1	0	1
0	1	1
0	0	0

Figure 3.15 indicates that it is not possible to separate the observations by a single linear decision boundary meaning this is a non-linearly separable problem. Perceptrons can only be used for linearly separable cases which implies it can not be used to solve this problem. However a combination of two perceptrons, that is two decision boundaries, may solve the problem. Let the first decision boundary separate the top right observation from the others (figure 3.16a) and the second decision boundary separate the bottom left observation from the others (figure 3.16b).

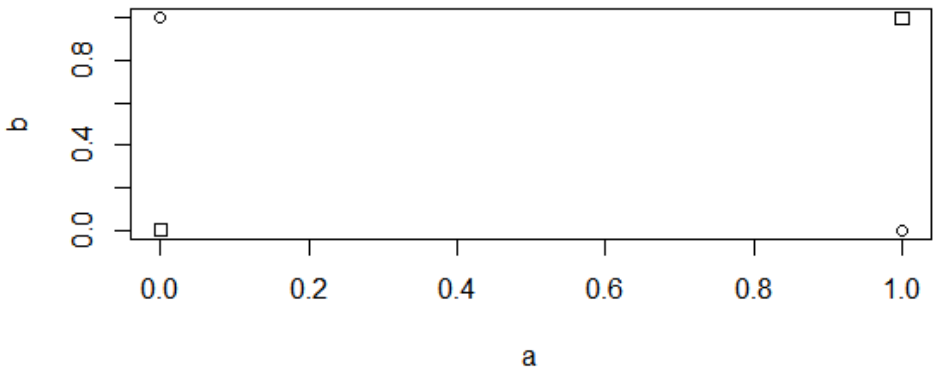


Figure 3.15: A graphical representation of the XOR problem.

Table 3.5a shows a learning task for the first perceptron which corresponds to the AND logical operation. Denoting the two classes as class zero and class one, the top right observation is assigned to class one and the rest of the observations are assigned to class zero. A normal vector of the decision boundary, that is the weight vector, could for example be $\begin{bmatrix} 1 & 1 \end{bmatrix}$ which makes the distance from the origin -1. Normalising the normal vector to length one results in $\mathbf{W}^1 = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$. Therefore a perceptron that can form this decision boundary has weight vector $\mathbf{W}^1 = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$ and bias $b_1 = -1$ and produces the decision boundary given by $\frac{1}{\sqrt{2}}X_1 + \frac{1}{\sqrt{2}}X_2 - 1 = 0$.

Table 3.5: The XOR-problem using two perceptrons.

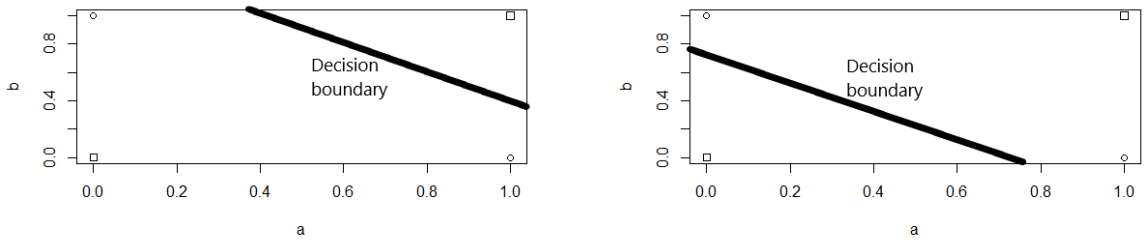
(a) Learning task for the first perceptron.

X_1	X_2	t
1	1	1
1	0	0
0	1	0
0	0	0

(b) Learning task for the second perceptron.

X_1	X_2	t
1	1	0
1	0	1
0	1	1
0	0	1

Table 3.5b shows a learning task for the second perceptron which corresponds to the OR logical operation. The bottom left observation is assigned to class zero and the rest of the observations are assigned to class one. A normal vector of the decision boundary, that is the weight vector, could for example be $\begin{bmatrix} 1 & 1 \end{bmatrix}$ which makes the distance from the origin $-\frac{1}{2}$. Normalising the normal vector to length one results in $\mathbf{W}^2 = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$. Therefore a perceptron that can form this decision boundary has weight vector $\mathbf{W}^2 = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$ and bias $b_2 = -\frac{1}{2}$ and produces the decision boundary is given by $\frac{1}{\sqrt{2}}X_1 + \frac{1}{\sqrt{2}}X_2 - \frac{1}{2} = 0$.



(a) XOR problem: The first decision boundary.

(b) XOR problem: The second decision boundary.

Figure 3.16: The XOR problem decision boundaries.

Combining the perceptrons results in a neural network with two layers which is trained using the learning task shown in table 3.6a. A new learning task, table 3.6b, can be formed which uses the outputs of this neural network as inputs for a perceptron with one neuron and the target values as solutions to the XOR logical operation which results in a linearly separable problem.

Table 3.6: The XOR-problem: Two neurons.

(a) Learning task of the neural network.

X_1	X_2	Y_1	Y_2
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

(b) Learning task of the outputs of the neural network.

Y_1	Y_2	t
1	1	0
0	1	1
0	1	1
0	0	0

A perceptron trained to learn this task could have weight vector $\mathbf{W}^3 = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$ and bias $b_3 = -\frac{1}{2}$. The final solution is a two layered neural network. This problem has demonstrated that multi-layered perceptron can be used to solve non-linearly separable patterns. This perceptrons can be trained to classify observations given a learning task. Thus suggests that there may be a way to learn the task given in table 3.4b without having to break it down. Consider a neural network used to solve the problem and suppose an input vector $\mathbf{X} = \begin{bmatrix} 1 & 1 \end{bmatrix}$ is passed which produces output $Z = 1$. This is an incorrect classification. The source of the misclassification is unclear as it could be coming from the weights or the intermediate outputs of the neural network. This is often referred to as the credit assignment problem as the error cannot be accredited to the hidden neurons (Jäger, 2005). The reason for this is that perceptrons use the hardlim function as the activation function. One way to avoid this is to use the identity function as the activation function which transforms the

neural network into a multi-layered adaline with mapping

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

for the hidden layer and mapping $Z = \begin{bmatrix} w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} + b_3$ for the output neuron. This dissolves into an adaline with no hidden layer since

$$\begin{aligned} Z &= \begin{bmatrix} w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} + b_3 = \begin{bmatrix} w_{31} & w_{32} \end{bmatrix} \left[\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right] + b_3 \\ &= \begin{bmatrix} w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} + b_3 \\ &= \begin{bmatrix} w_{31}w_{11} + w_{32}w_{21} & w_{31}w_{12} + w_{32}w_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + [w_{31}b_1 + w_{32}b_2 + b_3] \\ &= \begin{bmatrix} w'_{31} & w'_{32} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + b'_3 \end{aligned}$$

Thus a multi-layered adaline can be transformed to a single layered neuron. However adalines can only classify linear patterns so the non-linearly separable patterns problem has not been solved. One thing that could be done is to consider using other activation functions that mimic the hardlim function. An example of such an activation function is the log-sigmoid function (logsig), $f(\mathbf{x}) = \frac{1}{1+e^{-\beta \mathbf{x}}}$ which approximates the hardlim function as $\beta \to \infty$. A feed-foward neural network with these activation functions is referred to as a multi-layered perceptron. Multi-layered perceptrons provide a means to approximate continuous functions and therefore solve arbitrary learning tasks (Riedmiller, 1994).

3.9 Backpropagation

In general, the problem discussed in the previous section is of the form $F : [0,1]^n \to \mathbb{R}^m$. This can be written as a vector of m real-valued component functions where for each function f_i a neural network can be found. Consider a multi-layered perceptron with one hidden layer and no biases. The learning task of the neural network consists of p samples with $n-$dimensional feature vectors and p target variables. In the forward pass an input $\mathbf{X}^j$ is presented to the neural network and the output $\mathbf{Z}^j$ is produced. The aim is to minimise the error function e_j by using the steepest-descent optimisation method as a learning rule. The error function e_j is defined by $e_j = \frac{1}{2}||\mathbf{Z}^j - \mathbf{t}^j||^2 = \frac{1}{2}((z_1^j - t_1^j)^2 + (z_2^j - t_2^j)^2 + \cdots + (z_m^j - t_m^j)^2)$ for each sample $j \in 1, 2, \ldots p$. At output neuron k, $Z_k = g(N_k) = g(v_{k1}Y_1 + v_{k2}Y_2 + \cdots + v_{kq}Y_q)$ where Y_l denotes the output of the hidden layer. The error function E at neuron k is defined by

$$E = \frac{1}{2}||Z_k - t_k||^2 = \frac{1}{2}||g\left(N_k\right) - t_k||^2 = \frac{1}{2}||g(v_1Y_1 + v_2Y_2 + \cdots + v_qY_q) - t_k||^2.$$

The gradient of E using the chain rule is given by

$$\frac{\partial E}{\partial v_{ki}} = \frac{\partial E}{\partial Z_k} \cdot \frac{\partial Z_k}{\partial N_k} \cdot \frac{\partial N_k}{\partial v_{ki}} = (Z_k - t_k) \cdot g'\left(N_k\right) \cdot Y_i.$$

Denote $\delta_k^{out} = (Z_k - t_k) \cdot g'\left(N_k\right)$. The steepest descent update rule for the weights leading to output neuron k is $v_{ki}^* = v_{ki} - \eta\delta_k^{out}Y_i$ for learning coefficient $\eta > 0$ and $i = 1, 2, \ldots, q$. This rule is similar to the delta rule with a little adjustment (Riedmiller, 1994). At hidden neuron l, $Y_l = f(N_l) = f(w_{l1}X_1 + w_{l2}X_2 + \cdots + w_{ln}X_n)$. The error function E at neuron l is defined by

$$E = \frac{1}{2}||Y_l - t_l||^2 = \frac{1}{2}||f\left(N_l\right) - t_l||^2 = \frac{1}{2}||f(w_{l1}X_1 + w_{l2}X_2 + \cdots + w_{ln}X_n) - t_l||^2.$$

The gradient of E using the chain rule is given by

$$\frac{\partial E}{\partial w_{li}} = \frac{\partial E}{\partial Y_l} \cdot \frac{\partial Y_l}{\partial N_l} \cdot \frac{\partial N_l}{\partial w_{li}}.$$

Similar to the output neuron, $\frac{\partial Y_l}{\partial N_l} = f'\left(N_l\right)$ and $\frac{\partial N_l}{\partial w_{li}} = X_i$. The error function, E, of the neural network for each sample is given by

$$E = \frac{1}{2}||Z - t||^2 = \frac{1}{2}((z_1 - t_1)^2 + (z_2 - t_2)^2 + \cdots + (z_m - t_m)^2)$$

where each output neuron k is defined by $Z_k = g(N_k) = g(v_{k1}Y_1 + v_{k2}Y_2 + \cdots + v_{kq}Y_q)$. Replace each Z_k by $g(N_k) = g(v_{k1}Y_1 + v_{k2}Y_2 + \cdots + v_{kq}Y_q)$. Computing $\frac{\partial E}{\partial Y_l}$ yields

$$\begin{aligned}
\frac{\partial E}{\partial Y_l} &= \frac{\partial E}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial N_1} \cdot \frac{\partial N_1}{\partial Y_1} + \frac{\partial E}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial N_2} \cdot \frac{\partial N_2}{\partial Y_2} + \cdots + \frac{\partial E}{\partial Z_m} \cdot \frac{\partial Z_m}{\partial N_m} \cdot \frac{\partial N_m}{\partial Y_l} \\
&= (Z_1 - t_1) \cdot g'\left(N_1\right) \cdot v_{1l} + (Z_2 - t_2) \cdot g'\left(N_2\right) \cdot v_{2l} + \cdots + (Z_m - t_m) \cdot g'\left(N_m\right) \cdot v_{ml} \\
&= \delta_1^{out} \cdot v_{1l} + \delta_2^{out} \cdot v_{2l} + \cdots + \delta_m^{out} \cdot v_{ml} = \sum_{r=1}^{m} \delta_r^{out} \cdot v_{rl}.
\end{aligned}$$

The gradient of E at the hidden neurons is given by

$$\frac{\partial E}{\partial w_{li}} = \left(\sum_{r=1}^{m} \delta_r^{out} \cdot v_{rl}\right) \cdot f'\left(N_l\right) \cdot X_i$$

Denote δ_l^{hidden} as $\delta_l^{hidden} = \left(\sum_{r=1}^{m} \delta_r^{out} \cdot v_{rl}\right) \cdot f'\left(N_l\right)$. The steepest descent update rule for the weights leading to hidden neuron l is $w_{li}^* = w_{li} - \eta\delta_l^{hidden}X_i$ for $\eta > 0$ and $i = 1, 2, \ldots, n$. Let e_l be the error signal at the hidden neuron l where $e_l = \delta_1^{out} \cdot v_{1l} + \delta_2^{out} \cdot v_{2l} + \cdots + \delta_m^{out} \cdot v_{ml}$. The values $\delta_1^{out}, \delta_2^{out}, \ldots, \delta_m^{out}$ are computed at the output neuron. This suggests that they somehow

travel from the output neurons back to the hidden neuron l through the connecting links where they are used to measure the size of the error at hidden neuron l thereby distributing the error in the entire network. This strategy is called backpropagation (Riedmiller, 1994). Backpropagation is the algorithmic way in which the outputs of a neural network are sent back to the input recursively. The following algorithm describes how the backpropagation strategy works for a multi-layered perception with c layers.

Algorithm 3.2 The backpropagation Algorithm.

- Initialise all the weight and biases by small random numbers;
- For each training sample:
  - propagate the sample vector through the layers and calculate $Y^0 = X$ and $Y^j = f_j\left(W^j Y^{j-1}\right)$ for $j = 1, 2, \ldots, c$;
  - calculate $\delta_k^c = (Y_k^c - t_k) \cdot f_c'(N_k^c)$, $\delta_k^j = \left(\sum_l \delta_l^{j+1} w_{lk}^{j+1}\right) \cdot f_j'(N_k^j)$;
  - update the weights by $w_k^j = w_k^j + \eta \delta_k^j Y^{j-1}$.
- Repeat until the error over all training samples is sufficiently small.

Backpropagation is an algorithm used in computing to learn from errors, for example misclassification and improve performance. Using this tool computers can keep guessing and get better and better at guessing like humans do at one particular task. Computers can build a network of a lot of small tasks and are very good at each small task. This results in a system that can do bigger things like drive a car (Riedmiller, 1994). Neural networks can generalise to objects or data it has not seen before. The computer guesses a value and use calculus to calculate the error via partial derivatives. The error is fixed to get a better guess and backpropagate again to find a more fine tuned error. This method of repeatedly guessing shows the recursiveness of backpropagation. To demonstrate the backpropagation algorithm consider a two-layered neural network. For the first layer the neural network has weights and biases

$$\mathbf{W}^1 = \begin{bmatrix} 0.1 & 0.2 \\ -0.1 & 0.5 \end{bmatrix}, \mathbf{b}^1 = \begin{bmatrix} -0.1 \\ 0.2 \end{bmatrix}$$

and for the second layer the neural network has weights and biases

$$W^2 = \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}, b^2 = 0.1.$$

Suppose input vector $\mathbf{X} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is passed to the neural network. For the forward pass the

net input of the first layer, $\mathbf{N}^1 = \mathbf{W}^1\mathbf{X} + \mathbf{b}^1$, is given by

$$\begin{bmatrix} N_1 \\ N_2 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 \\ -0.1 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.6 \end{bmatrix}$$

and the output of the first layer is

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = logsig\left(\begin{bmatrix} N_1 \\ N_2 \end{bmatrix}\right) = logsig\left(\begin{bmatrix} 0.2 \\ 0.6 \end{bmatrix}\right) = \begin{bmatrix} 0.5498 \\ 0.6457 \end{bmatrix}$$

where *logsig* denotes the activation function which is the log-sigmoid function. The net input of the second layer, $N^2 = \mathbf{W}^{2'}\mathbf{Y} + \mathbf{b}^2$, is given by

$$N^2 = \begin{bmatrix} 0.2 & -0.1 \end{bmatrix} \begin{bmatrix} 0.5498 \\ 0.6457 \end{bmatrix} + 0.1 = 0.1454$$

and the output of the second layer is $Z = 0.1454$ since the activation function is the purelin function. For the backward pass the deltas are computed as

$$\delta^{out} = Z - 0 = 0.1454$$

for the outer layer and

$$\begin{aligned} \delta_1^{hidden} &= \delta^{out} \cdot 0.2 \cdot Y_1\,(1 - Y_1) = 0.007198, \\ \delta_2^{hidden} &= \delta^{out} \cdot -0.1 \cdot Y_2\,(1 - Y_2) = -0.003326 \end{aligned}$$

for the hidden layer. Using $\eta = 0.1$ as the learning rate, the weights which include the biases are updated as follows:

$$\mathbf{W}^2 = \begin{bmatrix} 0.2 \\ -0.1 \\ 0.1 \end{bmatrix} - 0.1 \cdot \delta^{out} \begin{bmatrix} Y_1 \\ Y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.1920 \\ -0.1094 \\ 0.08546 \end{bmatrix}$$

for the outer layer and

$$\begin{bmatrix} 0.1 \\ 0.2 \\ -0.1 \end{bmatrix} - 0.1 \cdot \delta_1^{hidden} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.09928 \\ 0.1993 \\ -0.1007 \end{bmatrix}$$

for the first neuron and

$$\begin{bmatrix} -0.1 \\ 0.5 \\ 0.2 \end{bmatrix} - 0.1 \cdot \delta_2^{hidden} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.09967 \\ 0.5003 \\ 0.2003 \end{bmatrix}$$

for the second neuron. Thus $\mathbf{W}^1 = \begin{bmatrix} 0.09928 & -0.09967 \\ 0.1993 & 0.5003 \\ -0.1007 & 0.2003 \end{bmatrix}$. After updating the weights the method is repeated until the error over all training samples is sufficiently small. The computations become more and more tedious hence computers are used to carry out the algorithm.

3.10 The Wine Data Neural Network

Consider the wine data set introduced in chapter 2. The goal is to build a neural network model that will correctly classify unseen observations into classes. The R code used to build the model is provided in appendix B.1 on page 127. The same data partition used in chapter 2, section 2.5 on page 26, was used to build and evaluate the performance of the neural network classifier.

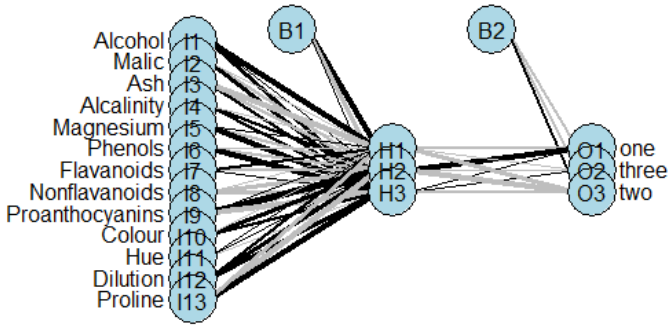


Figure 3.17: A neural network for the wine data set.

Figure 3.17 illustrates the neural network model built using the wine data set. Table 3.7 summarises the accuracies of this neural network. The test accuracy was 96.15%, with two observations belonging to class two being incorrectly classified to class three by the model. Repeated 7-fold cross validation increased the accuracy to 98.14%. The training and test accuracy rates of the neural network classifier built using the original data set are higher than that of the classification tree classifier built in chapter 2, section 2.5 on page 26. The accuracy of the neural network model built using repeated 7-fold cross validation is more than 12% higher than that of the classification tree model, see table 2.3 on page 30. For the wine data set the neural network model performs better than the classification tree model in correctly classifying observations. Detailed results of the model are available in

appendix B.2 on page 129. These results demonstrate how powerful and accurate a simple neural network model can be when used to solve classification problems.

Table 3.7: Accuracy of the wine data neural network.

Train accuracy	Test accuracy	Cross validation accuracy
1.0000	0.9615	0.9814

Chapter 4

Support Vector Machines

Support vector machines (SVM) are a supervised learning model used mostly as an approach to classification (James et al., 2013, pg 337). SVMs were introduced by Vapnik and colleagues (Cortes & Vapnik, 1995) and have, since the early 1990's, become one of the most favoured modeling techniques because they perform well in a variety of settings (James et al., 2013, pg 337). SVM were developed in four major steps. Initially they were introduced as a new learning machine for two-group or binary classification problems with linearly separable data known as maximum margin classifiers (Zhang, 2011). Secondly kernel functions were incorporated into the maximum margin classifiers and their formulation became similar to that of the current SVM (Zhang, 2011). Thirdly support vector classifiers were added to the SVM family which allow some observation to be on the incorrect side of the margin to cater for non-linearly separable training sets (Zhang, 2011). Lastly support vector classifiers were extended to support vector machines which result from enlarging the feature space in a specific way using kernels (James et al., 2013, 350). This chapter focuses on the development of SVM and is introduced by way of the well known iris data set.

4.1 The Iris Data Set

The Iris data set includes the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of three species of Iris. The plant species are *setosa*, *versicolor*, and *virginica*. For the sake of simplicity, this example will only consider flowers of the *setosa* species and the *versicolor* species and will only consider the measurements of the sepal length and width for those species. Table C.1 in appendix C.1 on page 133 shows the first ten observations of the subsetted data set. The goal is predict the species of flower based on the sepal length and width. Figure 4.1 is a scatter plot of these data. It is clear from this plot that these data are linearly separable by, for example, the straight line from the point 2.25,4 (approximately) to the point 4.5,7 (approximately).

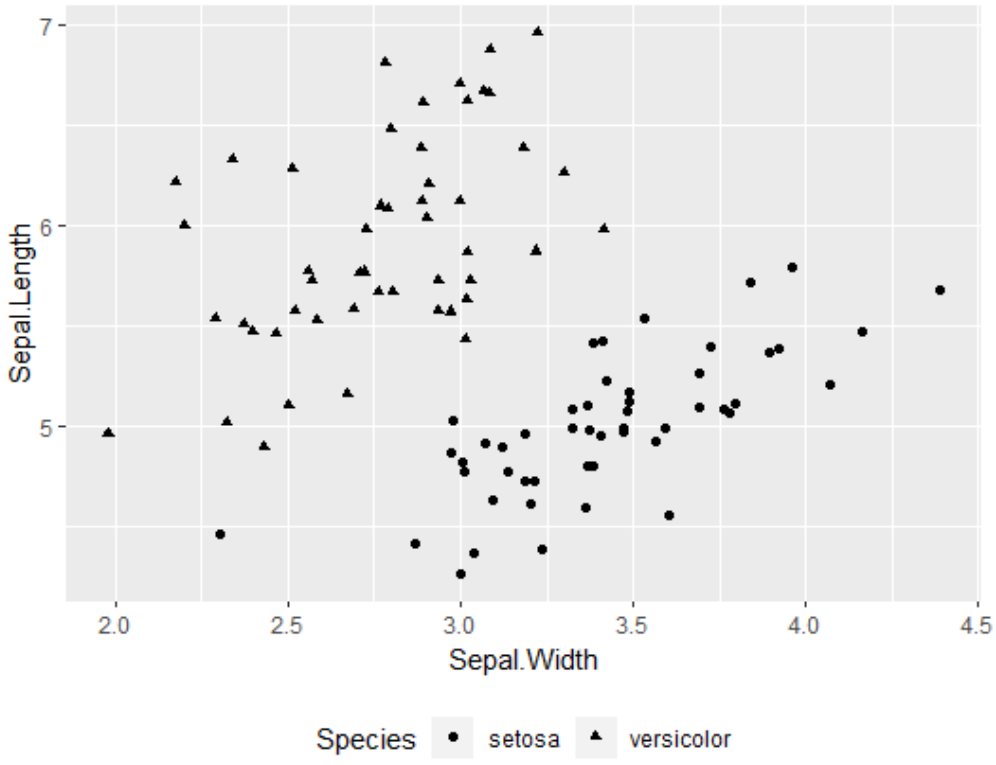
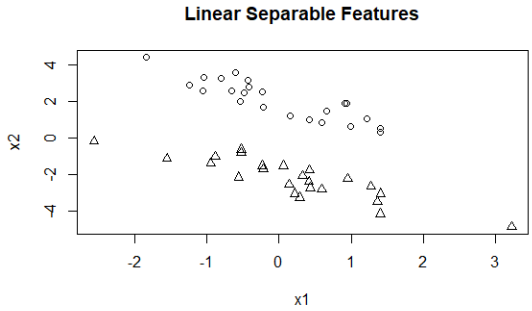


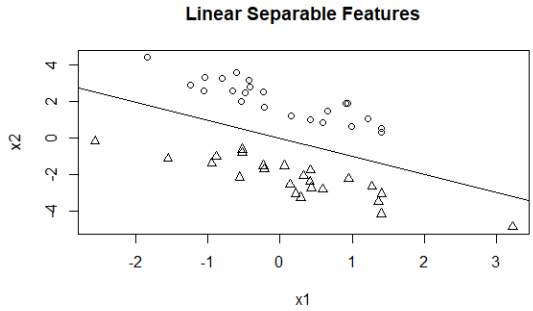
Figure 4.1: Scatter plot of the subsetted Iris data.

4.2 Linear Separability

Consider an input variable of n observations denoted by $\mathbf{X}$. The observations of $\mathbf{X}$ belong to one of two classes. The goal is to find a decision boundary that separates the observations into their respective classes. A decision boundary is a surface that separates an input vector space into two sets, one for each class. As an example consider the data represented in figure 4.2a. In this figure it is clear that the observations represented by $\triangle$ can be separated from the observations represented by $\bigcirc$ by a straight line for example as shown in figure 4.2b. The fact that the observations of $\mathbf{X}$ can be separated into the two groups by a straight line means that $\mathbf{X}$ is linearly separable. Linearly separable data refers to data that can be separated by a linear decision surface (Zaki et al., 2014, pg 571). In this example the linear decision surface is in the form of a straight line.



(a) Linearly separable data.

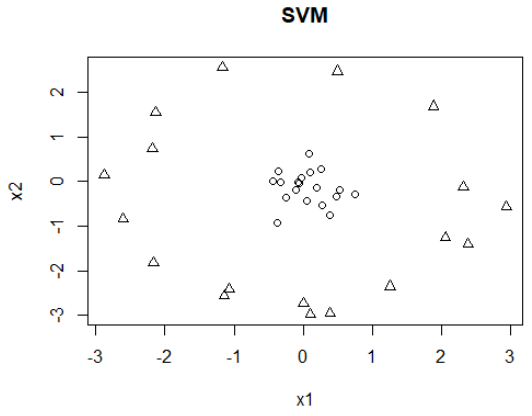


(b) Decision boundary for linearly separable data.

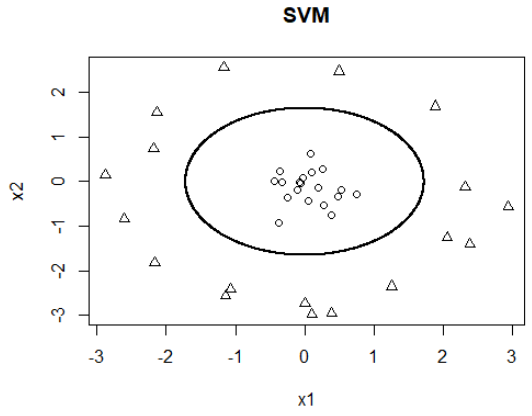
Figure 4.2: Linear separability.

4.3 Non-Linear Separability

Sometimes it is not possible to separate the observations of $\mathbf{X}$ by a linear decision boundary. Consider figure 4.3a. Unlike the observations in figure 4.2a, it does not seem as though the observations of $\mathbf{X}$ in figure 4.3a can be separated by a single linear decision surface. Data that can not be separated by a linear decision surface is referred to as non-linearly separable data (Izenman, 2008, pg 376). In figure 4.3b an alternative non-linear decision boundary, the ellipse, successfully separates the observations into the classes is shown.



(a) Non-linearly separable data.

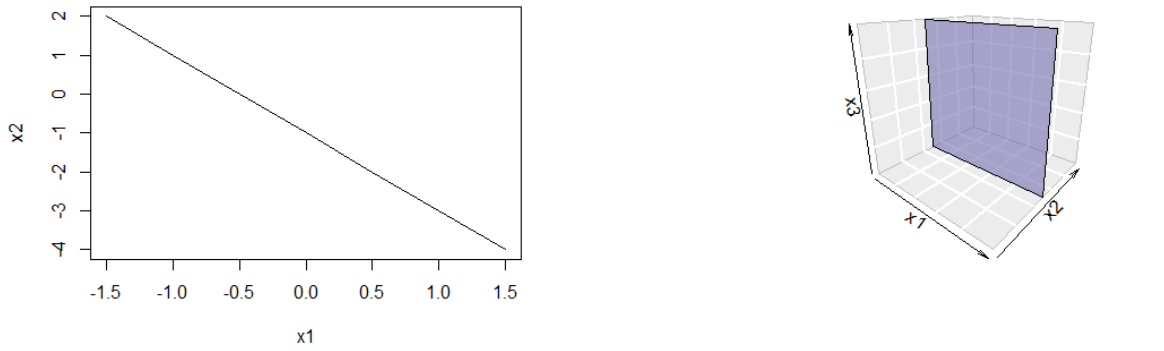


(b) Decision boundary for non-linearly separable data.

Figure 4.3: Non-linear separability.

4.4 The Maximal Margin Classifier

Maximal margin classifiers (MMC) determine a decision rule which separates data using a maximal margin (Franc & Hlaváč, 2003). In a p-dimensional space a hyperplane is a flat affine linear $(p-1)$-dimensional subspace (James et al., 2013, pg 338). An affine subspace is a subspace that does not necessarily pass through the origin. In two dimensions a hyperplane can be defined by the equation $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$ for the parameters β_0, β_1 and β_2, which is the equation of a straight line. Figure 4.4a shows a hyperplane in two dimensions defined by the equation $1 + 2X_1 + X_2 = 0$. This means that if there exists any point $\mathbf{X} = (X_1, X_2)'$ such that $1 + 2X_1 + X_2 = 0$ holds, then the point is on the hyperplane. In three dimensions a hyperplane can be defined by the equation $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 = 0$, for the parameters β_0, β_1, β_2 and β_3, which is the equation of a linear 3-dimensional surface. Figure 4.4b shows a hyperplane in 3 dimensions. This means that if there exists any point $\mathbf{X} = (X_1, X_2, X_3)'$ such that $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 = 0$ holds, then the point is on the hyperplane.

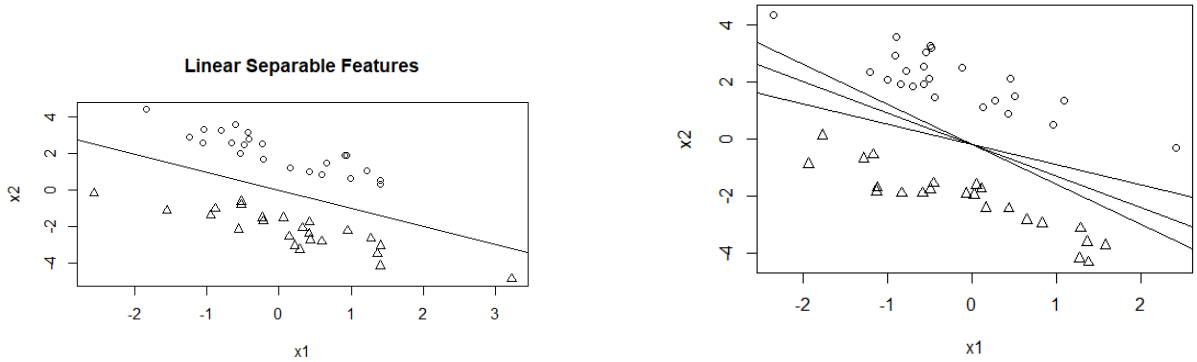


(a) A hyperplane in two dimensions.

(b) A hyperplane in three dimensions.

Figure 4.4: Examples of hyperplanes.

More formally a hyperplane in a p-dimensional space is defined by the equation $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = 0$ for the parameters $\beta_0, \beta_1, ..., \beta_p$ (James et al., 2013, pg 338). If there exists a point $\mathbf{X} = (X_1, X_2, ..., X_p)'$ such that the equation holds, then the point is on the hyperplane. Suppose that $\mathbf{X}$ is not a point on the hyperplane, that is $\mathbf{X}$ is on one side of the hyperplane. This can be represented by $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p > 0$ or $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p < 0$. Thus a hyperplane can be used as a rule dividing the p-dimensional space into two parts or components. This is illustrated in two dimensions in figure 4.5a. The feature space $\mathbf{X} = (X_1, X_2)'$ consists of observations that belong to one of two classes, denoted by triangles or circles, separated by a two dimensional hyperplane. None of the observations are points on the hyperplane. The circles represent the instance where $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$ and the triangles represent the instance where $\beta_0 + \beta_1 X_1 + \beta_2 X_2 < 0$. The hyperplane has successfully separated the observations into the two classes.



(a) Hyperplane in two dimensions.

(b) Hyperplanes in two dimensions.

Figure 4.5: Examples of hyperplanes.

Consider $\mathbf{X}$, n observations on a p-dimensional feature space, and suppose that the outcome variable $\mathbf{Y}$ is a known qualitative response belonging to one of two classes, denoted by $y_1, y_2, ..., y_n \in \{-1, 1\}$. The goal is to develop a classifier that will use the feature measurements $\mathbf{X}$ to classify the observations into the two classes. Assuming the data is linearly separable, a hyperplane can be used to separate the observations into the two classes (James et al., 2013, pg 339). If there exists a separating hyperplane for the data set then there will be an infinite number of such hyperplanes. This is because any given separating hyperplane can be shifted a little up or down, or rotated without coming into contact with any of the observations (James et al., 2013, pg 241). Figure 4.5b shows how shifting the hyperplane in figure 4.5a results in a second, third and more separating hyperplanes.

A classification rule may be constructed using a separating hyperplane (Franc & Hlaváč, 2003). There is one separating hyperplane which maximises the distance between the hyperplane and the observations. This hyperplane is called the maximal margin hyperplane or the optimal separating hyperplane (James et al., 2013, pg 341). The maximal margin hyperplane classifies all data and maximises the minimal distance, also referred to as the margin, between the data and the hyperplane (Gentile, 2001).

The margin is constructed by calculating the euclidean distance from each observation to a given separating hyperplane (James et al., 2013, pg 341). In figure 4.6 the maximal margin classifier is shown. The dotted lines represent the edges of the maximal margin. New observations are classified based on which side of the maximal margin hyperplane they belong to and this is known as the maximal margin classifier (MMC). Observations which lie on the edge of the margin are called support vectors. In figure 4.6 the support vectors are shaded in black. The maximal margin hyperplane depends directly on support vectors since any change, or movement, of these vectors will result in a shift of the maximal margin hyperplane whereas any movement of the other observations will not affect the separating hyperplane.

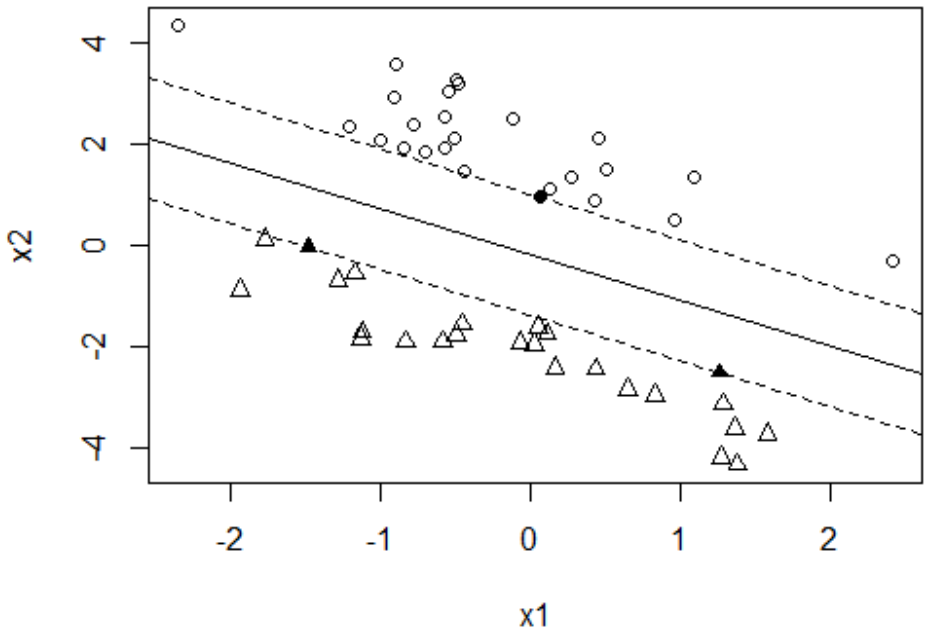


Figure 4.6: A maximal margin hyperplane in two dimensions.

As per Hastie et al. (2001, pg 371) and Mammone et al. (2009a), consider the construction of a MMC using a set of n linearly separable training observations on a two dimensional space, $\mathbf{x}_1 = \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix}, ..., \mathbf{x}_n = \begin{pmatrix} x_{n1} \\ x_{n2} \end{pmatrix}$, and associated class labels $y_1, ..., y_n \in \{-1, 1\}$. The MMC is based on the class of hyperplanes $\beta_0 + \mathbf{x}'\boldsymbol{\beta} = 0$, where $\boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}$, $\mathbf{x} \in \mathbb{R}^2$ and $\beta_0 \in \mathbb{R}$, corresponding to decision the function $f(\mathbf{x}) = sign(\beta_0 + \mathbf{x}'\boldsymbol{\beta})$. The points $\mathbf{x}$ which lie on the hyperplane satisfy $f(\mathbf{x}) = \beta_0 + \mathbf{x}'\boldsymbol{\beta} = 0$. $\boldsymbol{\beta}$ defines a direction perpendicular or normal to the hyperplane. Changing the value of β_0 shifts the hyperplane parallel to itself. Figure 4.7 illustrates a two dimensional hyperplane defined by the straight line $f(\mathbf{x}) = \beta_0 + \mathbf{x}'\boldsymbol{\beta} = 0$ with the following properties (Hastie et al., 2001, pg 105):

1. For any two points $\mathbf{x}_1$ and $\mathbf{x}_2$ lying on the hyperplane, $(\mathbf{x}_1 - \mathbf{x}_2)'\boldsymbol{\beta} = 0$;

2. $\boldsymbol{\beta}^* = \frac{\boldsymbol{\beta}}{\|\boldsymbol{\beta}\|}$ is the vector, of length one, which is normal to the hyperplane;

3. For any point $\mathbf{x}_0$ on the hyperplane, $\beta_0 + \mathbf{x}_0'\boldsymbol{\beta} = 0$, which is equivalent to $\mathbf{x}_0'\boldsymbol{\beta} = -\beta_0$;

4. The signed distance of any point $\mathbf{x}$ to the hyperplane is given by thelog of $(\mathbf{x} - \mathbf{x}_0)'\boldsymbol{\beta}^*$ where

$$
\begin{aligned}
(\mathbf{x}-\mathbf{x}_0)'\,\boldsymbol{\beta}^* &= (\mathbf{x}-\mathbf{x}_0)'\,\frac{\boldsymbol{\beta}}{\parallel \boldsymbol{\beta} \parallel} \text{ since } \boldsymbol{\beta}^* = \frac{\boldsymbol{\beta}}{\parallel \boldsymbol{\beta} \parallel} \\
&= (\mathbf{x}'\boldsymbol{\beta} - \mathbf{x}_0'\boldsymbol{\beta})\,\frac{1}{\parallel \boldsymbol{\beta} \parallel} \\
&= (\mathbf{x}'\boldsymbol{\beta} + \beta_0)\,\frac{1}{\parallel \boldsymbol{\beta} \parallel} \text{ since } x_0'\boldsymbol{\beta} = -\beta_0 \\
&= f(\mathbf{x})\,\frac{1}{\parallel f'(\mathbf{x}) \parallel}.
\end{aligned}
$$

Thus $f(\mathbf{x})$ is proportional to the signed distance from $\mathbf{x}$ to the hyperplane defined by $f(\mathbf{x}) = \beta_0 + \mathbf{x}'\boldsymbol{\beta} = 0$.

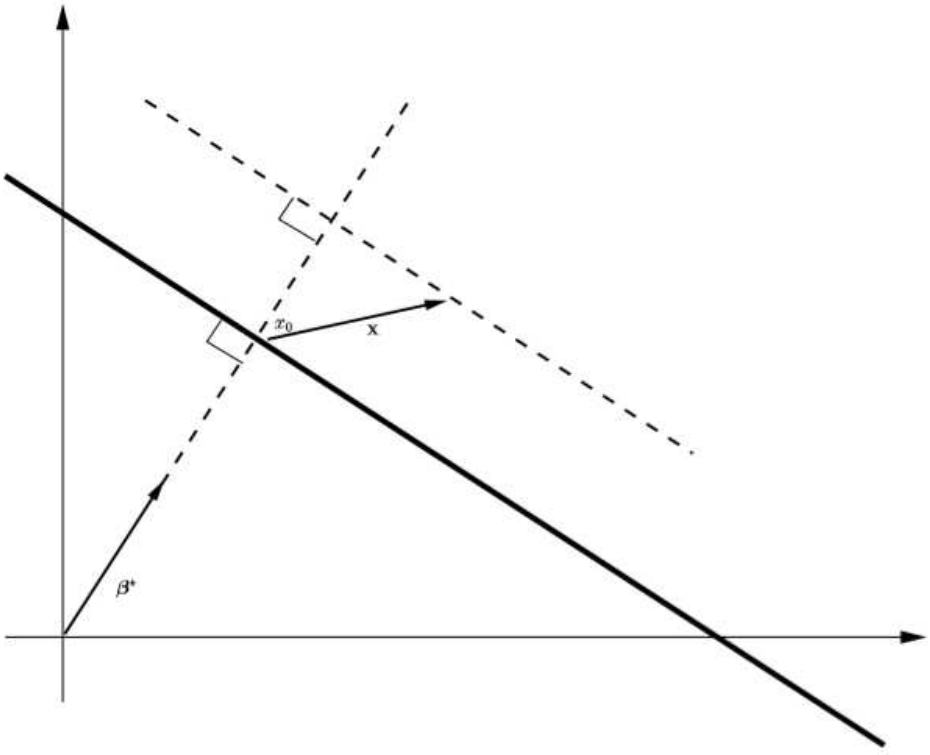


Figure 4.7: The linear algebra of a hyperplane.

Assume that on the edge of the margin the decision function has value ± 1. Consider two points, denoted by $\mathbf{x}_+$ and $\mathbf{x}_-$, that fall on the edge of the margin and belong to the positive and negative class respectively, that is $\beta_0 + \mathbf{x}_+'\boldsymbol{\beta} = 1$ and $\beta_0 + \mathbf{x}_-'\boldsymbol{\beta} = -1$. If an observation is known to belong to the positive class, that is $y_i = +1$, then $\beta_0 + \mathbf{x}_+'\boldsymbol{\beta} \geq 1$ and hence $y_i\left(\beta_0 + \mathbf{x}_+'\boldsymbol{\beta}\right) \geq 1$. If an observation is known to belong to the negative class, that is $y_i = -1$, then $\beta_0 + x_-'\boldsymbol{\beta} \leq -1$ and hence $y_i\left(\beta_0 + \mathbf{x}_+'\boldsymbol{\beta}\right) \geq 1$. Thus the same equation is obtained for both the positive and negative classes and hence generalising for both the positive and negative classes for any $\mathbf{x}$ yields

$$y_i\left(\beta_0 + \mathbf{x}'\boldsymbol{\beta}\right) - 1 \geq 0. \tag{4.1}$$

The observations that lie on the edge of the margin, namely those $\mathbf{x}$ such that

$$y_i\left(\beta_0 + \mathbf{x}'\boldsymbol{\beta}\right) - 1 = 0$$

are called the support vectors. The width of the margin is the euclidean distance between the support vectors belonging to the positive class and the support vectors belonging to the negative class. The margin is given by

$$\begin{aligned}\left(\mathbf{x}_+ - \mathbf{x}_-\right)'\boldsymbol{\beta}^* &= \left((1-\beta_0)\boldsymbol{\beta}^{-1} - (-1+\beta_0)\boldsymbol{\beta}^{-1}\right)\frac{\boldsymbol{\beta}}{\parallel \boldsymbol{\beta} \parallel} \\ &= \left((1-\beta_0) - (-1+\beta_0)\right)\boldsymbol{\beta}^{-1}\frac{\boldsymbol{\beta}}{\parallel \boldsymbol{\beta} \parallel}\end{aligned}$$

and hence

$$\left(\mathbf{x}_+ - \mathbf{x}_-\right)'\boldsymbol{\beta}^* = \frac{2}{\parallel \boldsymbol{\beta} \parallel} \tag{4.2}$$

Equation 4.2 shows that maximising the margin is equivalent to the maximisation of $\frac{2}{\|\boldsymbol{\beta}\|}$ subject to equation 4.1. This can be reduced to the maximisation of $\frac{1}{\|\boldsymbol{\beta}\|}$ or equivalently the minimisation of $\|\boldsymbol{\beta}\|$ or the minimisation of $\frac{1}{2}\|\boldsymbol{\beta}\|^2$ which is more mathematically convenient. This problem can be solved using Lagrange multipliers:

$$L = \frac{1}{2} \parallel \boldsymbol{\beta} \parallel^2 - \sum_{i=1}^{n} \alpha_i \left[y_i\left(\beta_0 + \mathbf{x}_i'\boldsymbol{\beta}\right) - 1\right] \tag{4.3}$$

where $\alpha_i \geq 0$, for $i = 1, ..., n$ are the Lagrange multipliers. Differentiating equation 4.3 with respect to $\boldsymbol{\beta}$ yields

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i = 0 \tag{4.4}$$

Differentiating equation 4.3 with respect to β_0 yields

$$\frac{\partial L}{\partial \beta_0} = \sum_{i=1}^{n} y_i\alpha_i = 0. \tag{4.5}$$

Solving equations 4.4 and 4.5 yields

$$\sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i = \boldsymbol{\beta} \tag{4.6}$$

$$\sum_{i=1}^{n} y_i\alpha_i = 0 \tag{4.7}$$

respectively. Substituting equations 4.6 and 4.7 back into equation 4.3 results in the following dual optimisation problem: Maxixmise L with respect to α_i subject to $\sum_{i=1}^{n} y_i\alpha_i = 0$, $\alpha_i \geq 0$, $i = 1, ..., n$.

$$
\begin{aligned}
L &= \frac{1}{2} \| \sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i \|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i \left(\beta_0 + \mathbf{x}_i' \sum_{i=1}^{n} y_k\alpha_k\mathbf{x}_k \right) - 1 \right] \\
&= \frac{1}{2} \left(\sqrt{\left(\sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i \right)^2} \right)^2 - \sum_{i=1}^{n} \alpha_i \left[\left(y_i\beta_0 + \sum_{i=1}^{n} y_iy_k\alpha_i\mathbf{x}_i'\mathbf{x}_k \right) - 1 \right] \\
&= \frac{1}{2} \left(\sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i \right)^2 - \sum_{i=1}^{n} y_i\alpha_i\beta_0 - \sum_{i=1}^{n}\sum_{k=1}^{n} y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k + \sum_{i=1}^{n} \alpha_i \\
&= \frac{1}{2} \left(\sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i' \right) \left(\sum_{i=1}^{n} y_k\alpha_k\mathbf{x}_k \right) - \beta_0 \sum_{i=1}^{n} y_i\alpha_i - \sum_{i=1}^{n}\sum_{k=1}^{n} y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k + \sum_{i=1}^{n} \alpha_i \\
&= \frac{1}{2} \left(\sum_{i=1}^{n}\sum_{k=1}^{n} y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k \right) - \beta_0 0 - \sum_{i=1}^{n}\sum_{k=1}^{n} y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k + \sum_{i=1}^{n} \alpha_i \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \left(\sum_{i=1}^{n}\sum_{k=1}^{n} y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k \right)
\end{aligned}
$$

Maximising L under the given constraints is a quadratic programming problem (Hastie et al., 2001, pg 373). A quadratic programming problem is a quadratic optimisation problem with linear constraints (Hastie et al., 2001, pg 373). If the data is linearly separable, the global optimal solution α_i, $i = 1, ..., n$, exists (Hastie et al., 2001, pg 374). For quadratic programming, the values of the primal and dual objective functions coincide at the optimal solutions if they exist. This is called zero duality gap.

Observations that are associated with positive α_i are support vectors for the positive class while observations with negative α_i are support vectors for the negative class. From equation 4.6 the decision function is given by $f(\mathbf{x}) = \beta_0 + \sum_{i \in S} y_i\alpha_i\mathbf{x}_i'\mathbf{x}$, where S is the set of support vector indices.

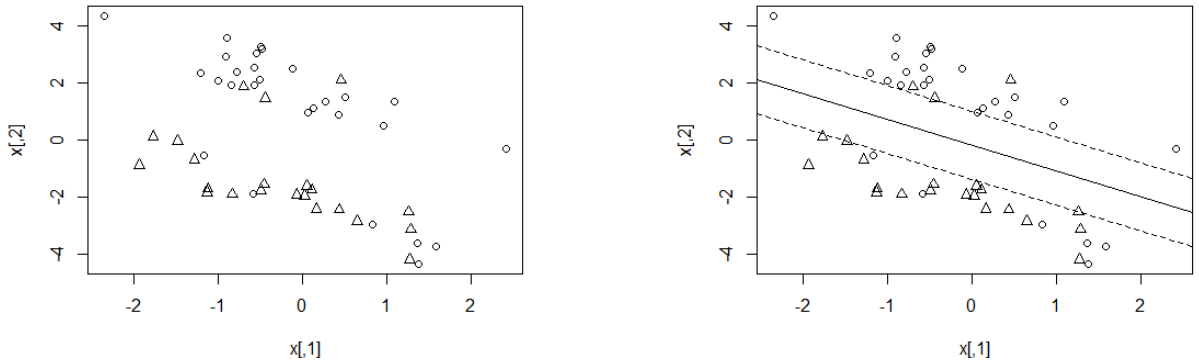
Often a separating hyperplane does not exist and as a result there is no MMC (James et al., 2013, pg 343). This problem required development of a hyperplane that almost separates the classes which lead to the establishment of what is known as the support vector classifier.

4.5 The Support Vector Classifier

Sometimes the observations of $\mathbf{X}$ are non-linearly separable and as a result cannot be perfectly separated by a hyperplane. It may be very difficult to find a MMC that can perfectly separate observations into two classes and if it does exist it may have a small margin (James et al.,

2013, pg 344). Ideally the margin should be large as the margin is the euclidean distance of an observation from the separating hyperplane which can be thought of as a measure of confidence that the observation was correctly classified (James et al., 2013, pg 344). The support vector classifier (SVC) was developed such that it did not necessarily perfectly separate the observations into two classes. The SVC, or the soft margin classifier, allows for a small proportion of the observations to be on the wrong side of the margin or the incorrect side of the hyperplane rather than finding the maximal margin such that all observations are not only on the correct side of the hyperplane but also on the correct side of the margin (James et al., 2013, pg 345).

Consider figure 4.8a where the observations of $\mathbf{X}$ belong to one of two classes denoted by circles and triangles. It is clear that there is no hyperplane that can perfectly separate the observations into the classes. In figure 4.8b the observations are separated by a hyperplane but some observations are on the incorrect side of the hyperplane. In this example six observations belonging to the class denoted by circles are on the incorrect side of the hyperplane and three of the observations belonging to the class denoted by triangles are on the incorrect side of the hyperplane. The two dimensional linear decision boundary in figure 4.8b is an example of a SVC since it allows some observations to violate the classification.



(a) Non-linearly separable data set. **(b)** Support vector classifier.

Figure 4.8: A data set that can not be separated by a hyperplane.

Consider a set of n non-linearly separable training observations on a two dimensional space with associated class labels $y_1, ..., y_n$. The construction of the SVC is similar to the construction of an MMC but allows some misclassifications on the training data. To obtain this, the constraints on equation 4.1 are relaxed but only when necessary. Slack variables, denoted by ξ_i, are introduced and allow individual observations to be on the incorrect side of the hyperplane. The slack variable ξ_i indicates where the i^{th} observation is located relative to the margin and hyperplane. If $\xi_i = 0$ then the i^{th} observation is on the correct side of the margin and hyperplane. If $0 < \xi_i < 1$ then the i^{th} observation is on the correct side of the hyperplane but on the incorrect side of the margin. If $\xi_i > 1$ then the i^{th} observation is on the

incorrect side of the hyperplane (Abe, 2005, pg 22). Introducing the slack variables changes the constraint in equation 4.1 to

$$y_i\left(\beta_0 + \mathbf{x}'\boldsymbol{\beta}\right) \geq 1 - \xi_i \tag{4.8}$$

$\xi_i \geq 0$, $\sum_{i=1}^{n} \xi_i <$ some constant $\forall i$. $\sum_{i=1}^{n} \xi_i$ is an upper bound on the number of training errors that are allowed. The optimisation problem to be solved is thus

$$\frac{1}{2} \parallel \boldsymbol{\beta} \parallel^2 + \gamma \sum_{i=1}^{n} \xi_i$$

subject to equation 4.8 where γ is a parameter chosen to penalise misclassification errors. A large γ corresponds to assigning a higher penalty to errors. Similar to the linearly separable case, introducing the Lagrange multipliers α_i and μ_i, results in

$$L = \frac{1}{2} \parallel \boldsymbol{\beta} \parallel^2 + \gamma \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \left[y_i\left(\beta_0 + \mathbf{x}_i'\boldsymbol{\beta}\right) - (1 - \xi_i)\right] - \sum_{i=1}^{n} \mu_i \xi_i \tag{4.9}$$

where α_i, $\mu_i \geq 0$, $i = 1, ..., n$. Differentiating equation 4.9 with respect to $\boldsymbol{\beta}$, β_0 and ξ_i yields

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i = 0 \tag{4.10}$$

$$\frac{\partial L}{\partial \beta_0} = \sum_{i=1}^{n} y_i \alpha_i = 0 \tag{4.11}$$

$$\frac{\partial L}{\partial \xi_i} = \gamma \sum_{i=1}^{n} - \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \mu_i = 0 \tag{4.12}$$

respectively. Solving equations 4.10, 4.11 and 4.12 results in

$$\sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i = \boldsymbol{\beta} \tag{4.13}$$

$$\sum_{i=1}^{n} y_i \alpha_i = 0 \tag{4.14}$$

$$\sum_{i=1}^{n} \alpha_i = \gamma \sum_{i=1}^{n} - \sum_{i=1}^{n} \mu_i \tag{4.15}$$

respectively. Substituting 4.13 and 4.14 into equation 4.9 results in the optimisation problem with linear constraints:

$$
\begin{aligned}
L &= \frac{1}{2}\Big|\Big|\sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i\Big|\Big|^2+\gamma\sum_{i=1}^{n}\xi_i-\sum_{i=1}^{n}\alpha_i\left[y_i\left(\beta_0+\mathbf{x}_i'\sum_{i=1}^{n}y_k\alpha_k\mathbf{x}_k\right)-(1-\xi_i)\right]-\sum_{i=1}^{n}\mu_i\xi_i \\
&= \frac{1}{2}\left(\sqrt{\left(\sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i\right)^2}\right)^2+\gamma\sum_{i=1}^{n}\xi_i-\sum_{i=1}^{n}\alpha_i\left[\left(y_i\beta_0+\sum_{i=1}^{n}y_iy_k\alpha_i\mathbf{x}_i'\mathbf{x}_k\right)-(1-\xi_i)\right]-\sum_{i=1}^{n}\mu_i\xi_i \\
&= \frac{1}{2}\left(\sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i\right)^2+\gamma\sum_{i=1}^{n}\xi_i-\sum_{i=1}^{n}y_i\alpha_i\beta_0-\sum_{i=1}^{n}\sum_{k=1}^{n}y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k+\sum_{i=1}^{n}\alpha_i(1-\xi_i)-\sum_{i=1}^{n}\mu_i\xi_i \\
&= \frac{1}{2}\left(\sum_{i=1}^{n} y_i\alpha_i\mathbf{x}_i'\right)\left(\sum_{i=1}^{n} y_k\alpha_k\mathbf{x}_k\right)-\beta_0\sum_{i=1}^{n}y_i\alpha_i-\sum_{i=1}^{n}\sum_{k=1}^{n}y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k+\gamma\sum_{i=1}^{n}\xi_i+ \\
&+\sum_{i=1}^{n}\alpha_i(1-\xi_i)-\sum_{i=1}^{n}\mu_i\xi_i \\
&= \frac{1}{2}\left(\sum_{i=1}^{n}\sum_{k=1}^{n}y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k\right)-\beta_00-\sum_{i=1}^{n}\sum_{k=1}^{n}y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k+\sum_{i=1}^{n}\alpha_i-\sum_{i=1}^{n}\alpha_i\xi_i+ \\
&+\left(\gamma\sum_{i=1}^{n}\xi_i-\sum_{i=1}^{n}\mu_i\xi_i\right) \\
&= \sum_{i=1}^{n}\alpha_i-\frac{1}{2}\left(\sum_{i=1}^{n}\sum_{k=1}^{n}y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k\right)-\sum_{i=1}^{n}\alpha_i\xi_i+\left(\sum_{i=1}^{n}\alpha_i\xi_i\right) \\
&= \sum_{i=1}^{n}\alpha_i-\frac{1}{2}\left(\sum_{i=1}^{n}\sum_{k=1}^{n}y_iy_k\alpha_i\alpha_k\mathbf{x}_i'\mathbf{x}_k\right)
\end{aligned}
$$

where L is maximised subject to $\sum_{i=1}^{n} y_i\alpha_i = 0$, $0 \leq \alpha_i \leq \gamma$, $i = 1, ..., n$. From equation 4.13 the decision function is given by $f(\mathbf{x}) = \beta_0 + \sum_{i\in S} y_i\alpha_i\mathbf{x}_i'\mathbf{x}$, where S is the set of support vector indices.

4.6 Support Vector Machines

The SVC separates data that is not linearly separable by a linear decision boundary. It allows for a few observations to be on the wrong side of the margin and/or on the wrong side of the hyperplane. Sometimes the SVC results in a large number of observations on the wrong side of the hyperplane. This defeats the goal of classification and hence other methods need to be considered. Figure 4.3a demonstrates observations that are non-linearly separable and would result in a relatively large number of misclassified observations if a linear decision boundary is used to separate them. The use of decision boundaries that are not linear is the generalisation of SVC to SVM. Figure 4.3b is an example of a non-linear decision boundary. The SVM used to classify the observations into one of the two classes in figure 4.9 results from enlarging

the feature space in a specific way using kernels (James et al., 2013, pg 350). In figure 4.9 a radial kernel was used and the resulting SVM successfully classified the observations into one of the two classes.

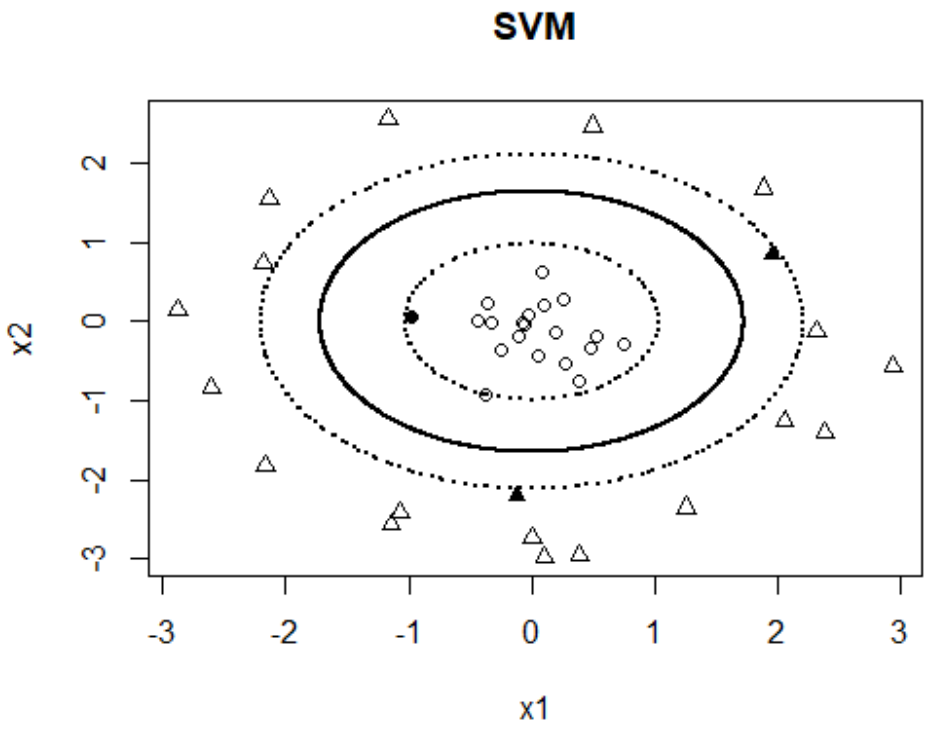


Figure 4.9: A SVM using a radial kernel.

In order to introduce a non-linear decision boundary between the classes suppose the original points, $\mathbf{x}_i$, in the feature space are mapped to points $\Phi(\mathbf{x}_i)$ in a high dimensional feature space, denoted H, using some non-linear transformation Φ (Zaki et al., 2014, pg 583). The transformed sample is then $\{\Phi(\mathbf{x}_i), y_i\}$ where y_i is the class of the observation. In sections 4.4 and 4.5, the described optimisation problems are in the form of inner product $\langle \mathbf{x}_i, \mathbf{x}_k \rangle = \mathbf{x}_i' \mathbf{x}_k$. When $\mathbf{x}_i$ is replaced by $\Phi(\mathbf{x}_i)$ in the construction of the model to accommodate non-linear decision boundaries then the data would only enter the optimisation problem by way of the inner product $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_k) \rangle = \Phi(\mathbf{x}_i)' \Phi(\mathbf{x}_k)$ (Izenman, 2008, pg 379). Computing the inner products in the high dimensional space, H, is very difficult and that is why Cortes & Vapnik (1995) introduced the kernel trick. The kernel trick is a less computationally expensive method used to compute inner products of the form $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_k) \rangle$ in feature space H. The idea behind the kernel trick is that instead of computing these inner products in H, which would be computationally expensive because of its high dimensionality, they are computed using a non-linear kernel function $K(\mathbf{x}_i, \mathbf{x}_k) = \Phi(\mathbf{x}_i)' \Phi(\mathbf{x}_k)$ in the original feature space. This trick speeds up the computations (Izenman, 2008, pg 379). In this manner a linear SVM is computed but where the computations are carried out in some other space.

4.6.1 Kernel Functions and their Properties

A kernel is a function K such that for all $\mathbf{x}_i$, $\mathbf{x}_k \in \mathbf{X}$,

$$K(\mathbf{x}_i, \mathbf{x}_k) = \Phi(\mathbf{x}_i)' \Phi(\mathbf{x}_k)$$

where Φ is the mapping $\Phi : \mathbf{x}_i \to \Phi(\mathbf{x}_i)$. The most important consequence of the kernel function is that the dimension of feature space H does not affect the computations of the inner products, in fact there is no need to know the explicit form of Φ (Mammone et al., 2009b). The only information used about the training sample is their kernel matrix defined as the square matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ such that $\mathbf{K}_{ik} = K(\mathbf{x}_i, \mathbf{x}_k)$ for a set of vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \subseteq \mathbf{X}$ and some kernel function K. The following are the properties of a kernel function which ensure it is a kernel for some feature space H:

- The function must be symmetric, that is

$$K(\mathbf{x}_i, \mathbf{x}_k) = \Phi(\mathbf{x}_i)' \Phi(\mathbf{x}_k) = \Phi(\mathbf{x}_k)' \Phi(\mathbf{x}_i) = K(\mathbf{x}_k, \mathbf{x}_i).$$

- The function must satisfy the Cauchy-Schwarz inequality, that is

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_k)^2 &= \left(\Phi(\mathbf{x}_i)' \Phi(\mathbf{x}_k)\right)^2 \\ &\leq \| \Phi(\mathbf{x}_i) \|^2 \| \Phi(\mathbf{x}_k) \|^2 \\ &\leq \left(\Phi(\mathbf{x}_i)' \Phi(\mathbf{x}_i)\right) \left(\Phi(\mathbf{x}_k)' \Phi(\mathbf{x}_k)\right) \\ &\leq K(\mathbf{x}_i, \mathbf{x}_i) K(\mathbf{x}_k, \mathbf{x}_k) \end{aligned}$$

- The matrix $\mathbf{K}_{ik} = K(\mathbf{x}_i, \mathbf{x}_k)$, $i, k = 1, \ldots, n$, must be positive semi-definite, that is the matrix must have non-negative eigenvalues for any set of data points $\mathbf{x}_i \in \mathbf{X}$, where $\mathbf{X}$ is a finite feature space.

The support vector classifier described in section 4.5 determines linear decision boundaries in the original feature space. To adapt the classifier such that it computes the linear decision boundary in feature space H using the kernel trick the optimisation problem in equation 4.9 and its solution can be adjusted such that the Lagrange function is of the form

$$L = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \left(\sum_{i=1}^{n} \sum_{k=1}^{n} y_i y_k \alpha_i \alpha_k \Phi(\mathbf{x}_i)' \Phi(\mathbf{x}_k) \right) \tag{4.16}$$

$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \left(\sum_{i=1}^{n} \sum_{k=1}^{n} y_i y_k \alpha_i \alpha_k K(\mathbf{x}_i, \mathbf{x}_k) \right) \tag{4.17}$$

The solution function, $f(\mathbf{x})$, is given by

$$\begin{aligned} f(\mathbf{x}) &= \beta_0 + \sum_{i \in S} y_i \alpha_i \Phi(\mathbf{x}_i)' \Phi(\mathbf{x}) \\ &= \beta_0 + \sum_{i \in S} y_i \alpha_i K(\mathbf{x}_i, \mathbf{x}) \end{aligned}$$

where L is maximised subject to $\sum_{i=1}^{n} y_i \alpha_i = 0$, $0 \leq \alpha_i \leq \gamma$, $i = 1, ..., n$.

4.6.2 Examples of Kernel Functions

Table 4.1 provides examples of commonly used kernel functions $K(\mathbf{x}_i, \mathbf{x}_k)$ (Karatzoglou et al., 2006), where $\alpha > 0$ is a scale parameter, γ is a parameter chosen to penalise misclassification errors, $c \geq 0$ and d is an integer. The polynomial and radial basis kernel functions provided in table 4.1 are used in the application chapter (see chapter 5 on page 75) of this thesis to develop support vector machine classifiers. More commonly used kernels are available in Karatzoglou et al. (2006).

Table 4.1: Examples of kernel functions.

Kernel	$K(\mathbf{x}_i, \mathbf{x}_k)$
Polynomial of degree d	$(\alpha(\mathbf{x}_i' \mathbf{x}_k) + c)^d$
Radial basis function	$\exp(-\gamma \parallel \mathbf{x}_i - \mathbf{x}_k \parallel^2)$
Sigmoid	$\tanh(\alpha(\mathbf{x}_i' \mathbf{x}_k) + c)$

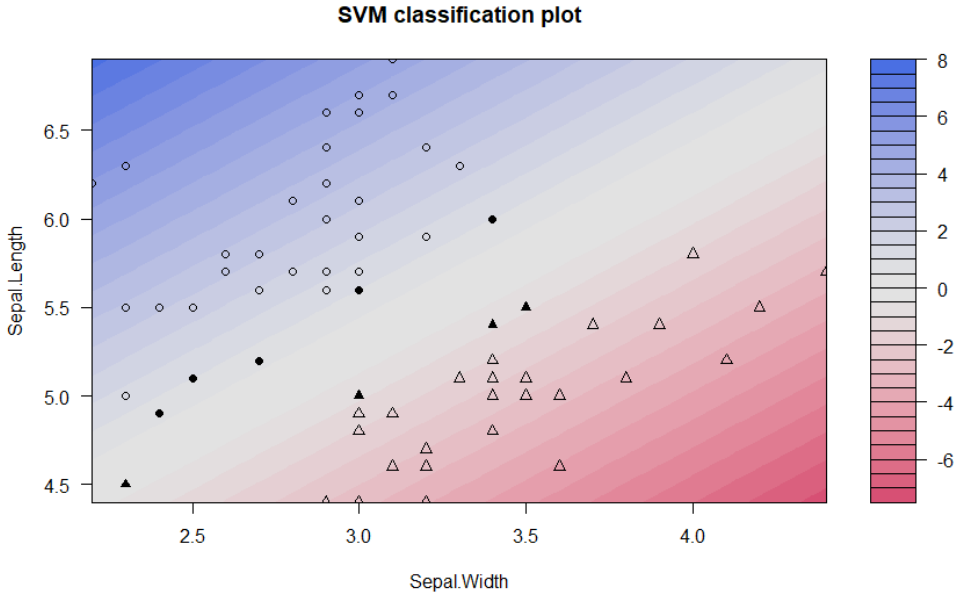
4.7 The Iris Data Support Vector Machine

Consider the subset of the Iris data set introduced in section 4.1 on page 59. In this section we fit a support vector machine classifier on these data and evaluate how well it classifies unseen observations. The data set was split into a train and test set using a 70/30 proportion, that is 70% of the data set was used to train the model and the rest was reserved as the test set. In section 4.1 it was established that these data are linearly separable and hence a linear kernel support vector machine was used to build the model. The term support vector machine is used loosely as, since the classes are separable by a linear boundary, a maximal margin classifier (see section 4.4) is used to determine the classification boundary. The R code used to build the model is provided in appendix C.2 on page 133. Table 4.2 provides the resulting confusion matrix for the test data after the support vector machine was trained on the training data. The classifier had a 100% test accuracy rate as it correctly classified all unseen observations, that is it correctly classified all observations from the test data set. Introducing repeated 7-fold cross validation during model training resulted in a slightly lower accuracy of 99.08%. A plot of the support vector machine classifier is shown in figure 4.10.

Points around zero are on the decision boundary. In section 4.4 support vectors are defined as observations which lie on the edge of the margin of the decision boundary. In figure 4.10 the support vectors are shaded in black to distinguish them from the other points. The classification resulted in nine support vectors. Detailed results of the model are available in appendix C.3 on page 135.

Table 4.2: The confusion matrix of the subsetted Iris data SVM.

Predicted	Actual	
	setosa	*versicolor*
setosa	15	0
versicolor	0	15



Train accuracy	Test accuracy	Cross validation accuracy
1.0000	1.0000	0.9908

Figure 4.10: Plot of the SVM: training data.

Chapter 5

Default in Payment on Credit Cards

Credit scoring is the practice of analysing a person's background and credit application in order to predict the risk associated with awarding the person credit (Mester, 1997). The purpose of this chapter is to develop a credit scoring system and compare the results of several classification models used to predict credit card payment default. Yeh & Lien (2009) utilised discriminant analysis, logistic regression and Bayes classifier to develop a credit scoring system. They used a data set generated from a bank in Taiwan which is available on the UCI Machine Learning Repository. The data set has thirty thousand (30 000) observations of the input vector, $\mathbf{X} = \begin{bmatrix} X_1 & X_2 & \cdots & X_{23} \end{bmatrix}'$, and a binary response $\mathbf{Y}$. If the client defaults on their credit card payment $Y = 1$, otherwise $Y = 0$. The input vector has twenty-three (23) feature variables, namely:

- X_1: The amount of credit given to the client which includes both their individual credit and their family, or supplementary, credit.
- X_2: The gender of the client. $X_2 = 1$ if the client is a male and $X_2 = 2$ if the client is a female.
- X_3: The level of education of the client. 1 = graduate school, 2 = university, 3 = high school and 4 = other.
- X_4: The marital status of the client. 1 = married, 2 = single and 3 = other.
- X_5: The age of the client in years.
- X_6 to X_{11}: The client's history of past monthly payment records from April to September in 2005 where X_6 = the repayment status in September, X_7 = the repayment status in August, ..., X_{11} = the repayment status in April. The measurement scale for the repayment status is: -1 = pay duly, 1 = payment delay for one month, 2 = payment delay for two months, ...,8 = payment delay for eight months, 9 = payment delay for nine months and above.

- X_{12} to X_{17}: The amount of the client's bill statement from April to September in 2005 where X_{12} = amount of bill statement in September, X_{13} = amount of bill statement in August, ..., X_{17} = amount of bill statement in April.
- X_{18} to X_{23}: The amount of the client's previous payment from April to September in 2005 where X_{18} = amount paid in September, X_{19} = amount paid in August, ..., X_{23} = amount paid in April.

Table D.1 in appendix D.1 on page 139 illustrates the first ten observations of this data set. The distribution of the classes is shown in table 5.1.

Table 5.1: Class distribution of credit data set.

Class (label)	Frequency	Percentage
No	23 364	77.88%
Yes	6 636	22.12%

The code for this analysis can be found in the appendix D. Out of the 30 000 observations in the data set 6 636 belong to the positive class, that is 22.12% of the clients default on their credit card payment. This suggest that the data set is imbalanced since the classes are not 50/50 distributed. Imbalanced data occurs when one class is represented by a large number of examples, while the other is represented by only a few (Batista et al., 2004). Imbalanced data hinders the performance of classification techniques particularly when the minority class is the positive class (Chawla et al., 2004), as in this case. The most common way to assess the performance of classifiers is using error and accuracy rates derived form the confusion matrix which are discussed in chapter 1, section 1.4. Since the data set is imbalanced the results given by these summaries may be unreliable because they are strongly biased to favour the largely represented class (Basheer & Hajmeer, 2000). As a result, alternative performance assessment techniques, namely specificity, sensitivity and AUC are reported.

There are nine (9) qualitative feature variables and fourteen (14) numeric feature variables in the credit data set. Binary numbers could have been introduced to represent the qualitative feature variables, for example if a feature variable is assigned to four levels then each level may be represented by two binary numbers such as 00, 01, 10 or 11 (Basheer & Hajmeer, 2000). The quantitative features in this data set are normalised prior to training. The qualitative feature variables are converted to numeric variables by the train function in R when fitting these classifiers. A summary of all variables is available in table D.2 in appendix D.2 on page 141. Most of the feature variables have outliers. As mentioned in chapter 2, section 2.1, good practice would be to remove any outliers as they might affect the training phase of the classifier. No outliers were removed for the sake of simplicity and to allow comparison to Yeh & Lien (2009) results. The credit data set has twenty-three (23) feature variables and as a result may be considered as a high dimensional data set. High dimensional data sets can sometimes have highly correlated variables and this hinders the learning phase of model development

and might affect the performance of the model. Figure 5.1 shows the correlations of the feature variables in the data set. Feature variables X_{12} to X_{17} are all strongly, positively correlated. These variables represent the amount of the client's bill statement from April to September in 2005. Not much correlation exists between most of the other feature variables. Dimension reduction also serves as a means to deal with a strong existing correlation between feature variables in a data set and may improve model performance.

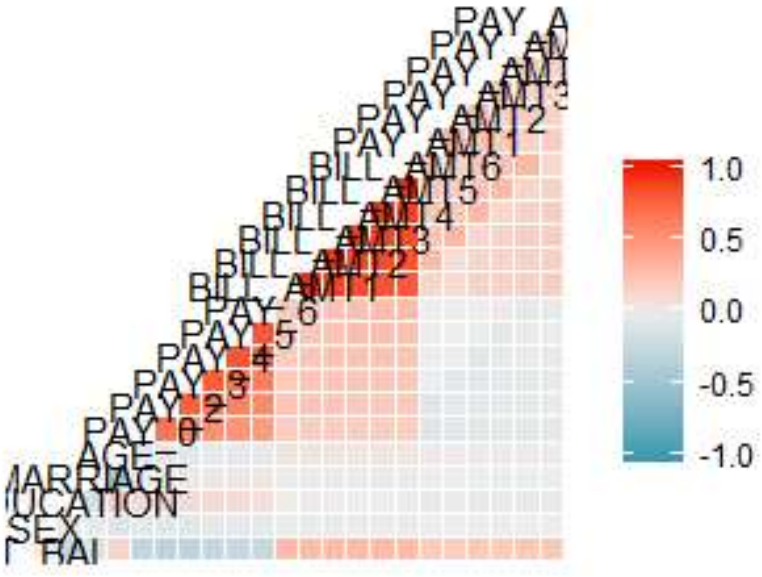


Figure 5.1: Variance-covariance diagram of the credit data feature variables.

The variance-covariance matrix of the credit data set feature variables is available in tables D.3 on page 142, D.4 on page 143 and D.5 on page 144 in appendix D.3. Two feature variables, age (X_5) and limit balance (X_1), were randomly chosen and plotted to visualise class distribution of these data (figure 5.2). It is clear that these data are not linearly separable over these variables. Figure 5.3 indicates that this observation would seem to hold true over all feature variables.

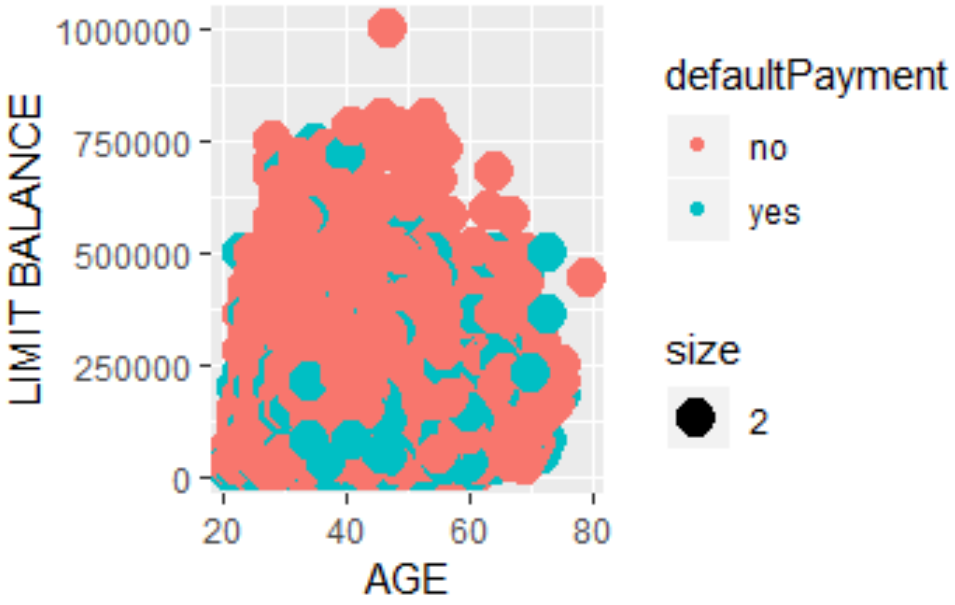
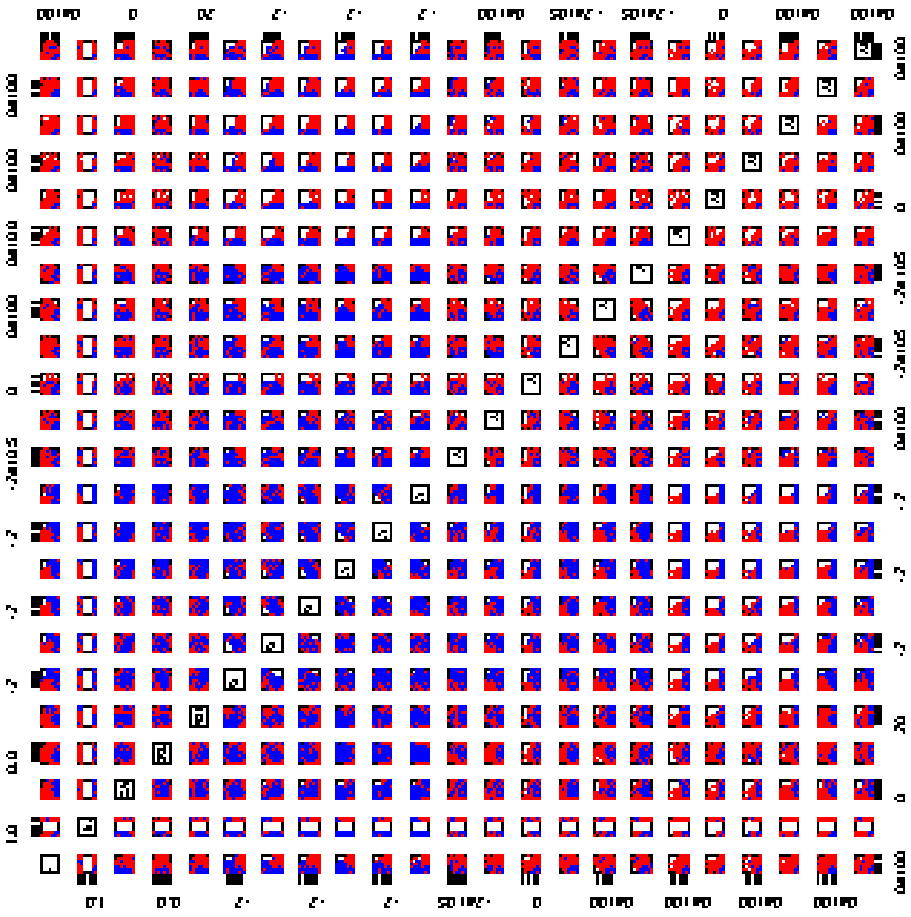


Figure 5.2: Class distribution resulting from the age and limit balance feature variables.



Red denotes "No" and blue denotes "Yes" to default payment.

Figure 5.3: Class distribution over all feature variables.

After data preprocessing the data set was split into a train data set and a test data set. An investigation was conducted to decide on suitable classifier parameters based on different proportions of data partitions. Basheer & Hajmeer (2000) state that there are no mathematical rules for the determination of the required sizes of the data subsets. May et al. (2010) discusses different data splitting approaches. The considered partitions were between a 70/30 proportion, that is 70% of the data set is used to train the model and 30% of the data set is used to test the performance of the model, a 80/20 proportion and lastly a 90/10 proportion. Following this repeated 7-fold cross validation was used to estimate the accuracy of the classifiers.

5.1 Artificial Neural Networks

After importing the data, various preprocessing methods were considered. The data set was checked to ensure that there was no missing data as neural network systems usually handle only complete input data cases. When cases are missing, various methods need to be employed to deal with those cases (Ennett et al., 2001). The data set in question has no missing data. Neural networks are not easy to train and tune using a raw data set (Jayalakshmi & Santhakumaran, 2011). An important practice is to normalise the data set before training a neural network. Avoiding normalisation may lead to useless results or to a very difficult training process where, in most cases, the algorithm will fail to converge before the number of maximum iterations is reached (Basheer & Hajmeer, 2000). The two most common methods used to scale the data sets are the z-normalisation and the min-max normalisation techniques (Angelini et al., 2008). Alternative normalisation techniques are discussed in Jayalakshmi & Santhakumaran (2011). The z-normalisation technique was used to scale the credit data. This data set contains univariate outliers. Using the min-max normalisation technique when there are outliers in the data set could result in the loss of useful information and could introduce factors that hinder model performance (Angelini et al., 2008). The scaled data set was divided into a training and test set.

For each train/test split, five neural network models were built where each neural network had between three (3) and seven (7) nodes. Table 5.2 provides the test accuracies of the models developed from the different train/test splits for the neural network classifiers composed of different number of nodes. The table shows that the larger the training data the higher the accuracy. In general, the accuracy results are not significantly different from each other. Varying the split proportion does not significantly impact the resulting model accuracy. However the 90/10 split gave the highest accuracy for three (3) of the five (5) models considered constructed using different data partitions.

Table 5.2: NN test accuracy rates for different data partitions number of nodes.

Number of nodes	Accuracy		
	70/30	80/20	90/10
3	0.8199	0.8215	0.8183
4	0.8198	0.8236	0.8246
5	0.8203	0.8218	0.8223
6	0.8173	0.8220	0.8243
7	0.8199	0.8218	0.8193

Deciding on the number of nodes in a neural network is not an exact science, therefore the trial and error approach plays a role in this process (Panchal & Panchal, 2014). One way to choose the most suitable number of nodes is to compare how the accuracy of the predictions change as the number of nodes is modified. The number of nodes considered in the credit scoring neural networks constructed above were between three and seven nodes. Neural network classifiers with 8,10,15,20,30 and 35 nodes were constructed using the 90/10 data partition. The accuracy rates of these neural networks are shown in figure 5.4. These computationally expensive neural networks do not perform significantly better than the models previously constructed. As a result only neural networks with between three and seven nodes will be considered for this study.

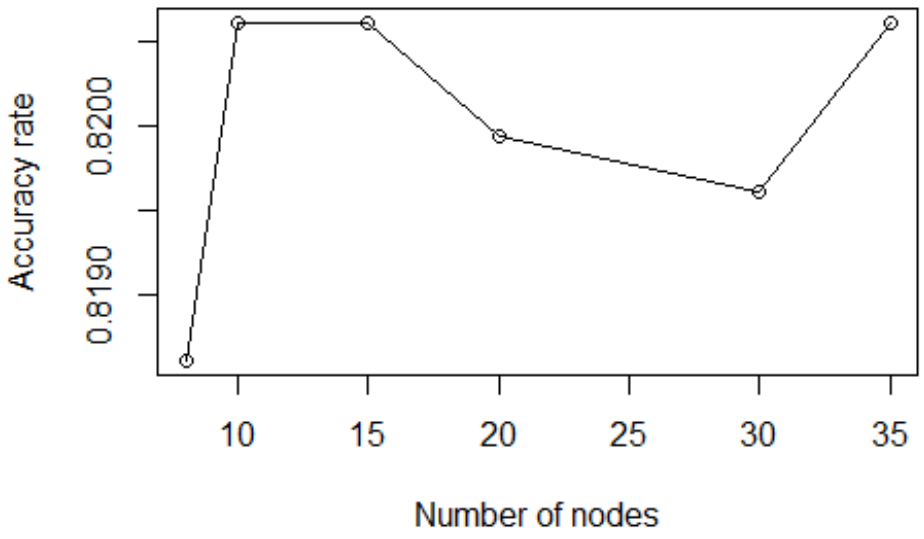


Figure 5.4: Test accuracy rates of neural networks with eight and more number of nodes.

Table 5.3 provides the results of the performance evaluation of five NNs built using a 90/10 data split. The first model was built using three nodes and yielded a test accuracy of 0.8183. The highest test accuracy came from the classifier with four nodes in its hidden layer. This

finding suggests that the number of nodes that should be used to build the model should be four but the accuracy rates of these models are not particularly different.

Table 5.3: 90/10 split neural network results with between three and seven nodes.

Number of nodes	Train accuracy	Test accuracy	Sensitivity	Specificity
3	0.8200	0.8183	0.9521	0.3469
4	0.8219	0.8246	0.9568	0.3590
5	0.8214	0.8223	0.9478	0.3801
6	0.8210	0.8243	0.9551	0.3635
7	0.8226	0.8193	0.9478	0.3665

Imbalance of data can hinder the classifier learning process and the accuracy performance metric of the model does not take this into consideration (see section 1.4 on page 4). To further assess the performance of these models ROC curves were used. Figure 5.5 shows the ROC curves of these five models and table 5.4 provides the AUC of these models. The ROC curves of the models are quite similar as are the AUC of these models. The neural network with seven nodes in its hidden layer was the best performing model with the highest AUC value of 0.7866. The AUC measure is not biased by imbalance data. This suggested that the neural network built using seven nodes should be chosen as the final model. The results of the models built using the 70/30 split and the 80/20 split indicated that in both cases the AUC of the neural network built using seven nodes was the highest (figure 5.4).

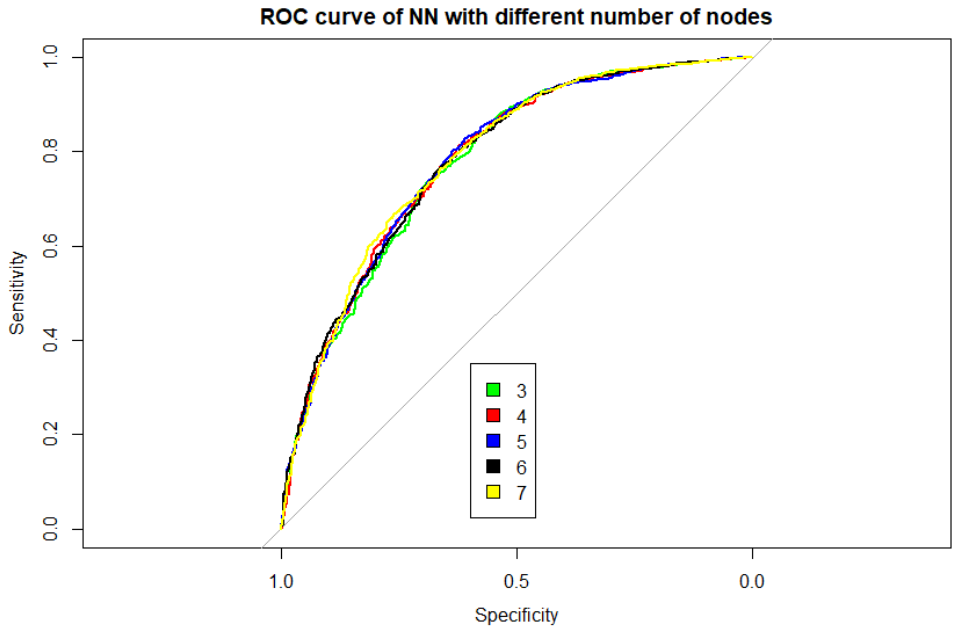


Figure 5.5: ROC curves of neural network models with a different number of nodes.

Table 5.4: AUC of the neural networks with between three and seven nodes.

Number of nodes	3	4	5	6	7
AUC of model (90/10)	0.7740	0.7802	0.7796	0.7815	0.7866
AUC of model (80/20)	0.7660	0.7702	0.7710	0.7720	0.7724
AUC of model (70/30)	0.7640	0.7660	0.7650	0.7665	0.7715

Choosing a final model based on the investigation conducted above is very naive. Good practice is to use cross validation to further evaluate the performance of the model with the varying parameter values in order to choose the optimal model. Repeated 7-fold cross validation was utilised using the R caret package. The output can be found in section D.5 on page 148. The nnet package used to build these models was used to construct a visualisation of this NN as shown in figure 5.7. The AUC is summarised in figure 5.6.

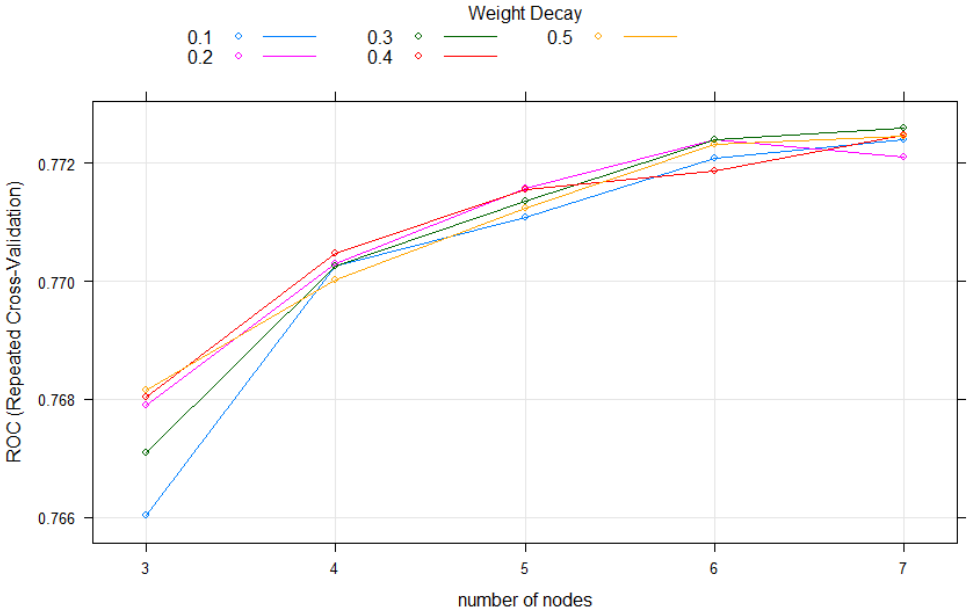
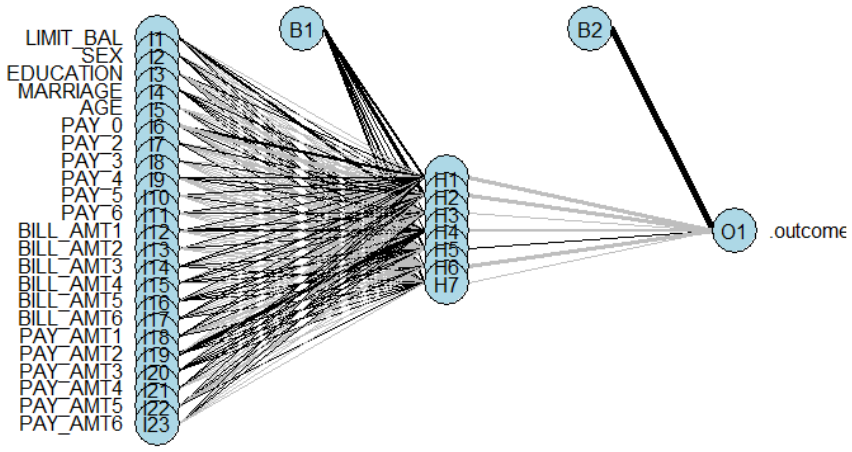


Figure 5.6: Cross validation ROC results for NN built with between three and seven nodes.

The cross validation results show that the optimal model has 7 nodes in its hidden layer and a weight decay of 0.3. Weight decay is a term added to the error function of a NN as part of a regularisation technique (Lauret et al., 2008). It penalises large values for the weights that are known to be responsible for excessive curvature in the model (Lauret et al., 2008). Regularisation is a technique used in modelling to avoid overfitting (Lauret et al., 2008).



Test accuracy	AUC	Sensitivity	Specificity
0.8243	0.7871	0.9521	0.3741

Figure 5.7: Performance of the NN credit scoring classifier.

5.2 Support Vector Machines

The same training set used to build the NN classifiers was used to build various SVM classifiers to enable model performance comparison. The difference between the SVM classifiers was the type of kernel used to build the models. As discussed in chapter 4 on page 59, the kernel determines the type of decision boundary that will be used to separate the data. The kernels considered here were the linear, radial and polynomial kernels. A linear kernel suggests that the data is linearly separable into classes. A radial basis kernel decision boundary is produced by projecting the data set into a higher-dimensional space using the kernel function $K(\mathbf{x}_i, \mathbf{x}_k) = \exp(-\gamma \parallel \mathbf{x}_i - \mathbf{x}_k \parallel^2)$. A polynomial kernel decision boundary is produced by projecting the data set into a higher-dimensional space using the kernel function $K(\mathbf{x}_i, \mathbf{x}_k) = (\alpha(\mathbf{x}_i'\mathbf{x}_k) + c)^d$ (see section 4.6 on page 70). $\alpha > 0$ is a scale parameter, γ is a parameter chosen to penalise misclassification errors, $c \geq 0$ and d is an integer (Izenman, 2008, pg 381). In order to choose the best classifier the performance of the different SVM models were evaluated. Table 5.5 provides a performance summary of these SVMs. While the SVM with a polynomial kernel had the highest sensitivity, the SVM with a radial kernel performed best as this SVM had the highest training and test accuracy.

Table 5.5: Summary of the SVM performance built with different kernels.

Kernel	Train accuracy	Test accuracy	Sensitivity	Specificity
Linear	0.8101	0.8119	0.9705	0.2505
Radial	0.8230	0.8213	0.9593	0.3348
Polynomial	0.8054	0.8126	0.9713	0.2534

ROC curves were constructed for these SVMs as a means to compare their performance. The ROC curve produced by the SVM with a radial kernel indicates that this is the best performing classifier. Table 5.6 provides the AUC values of these SMVs. The AUC value for the radial kernel SVM is the highest. This information combined with evaluating the accuracy rates of these classifiers confirms that the final SVM model should be build with a radial kernel.

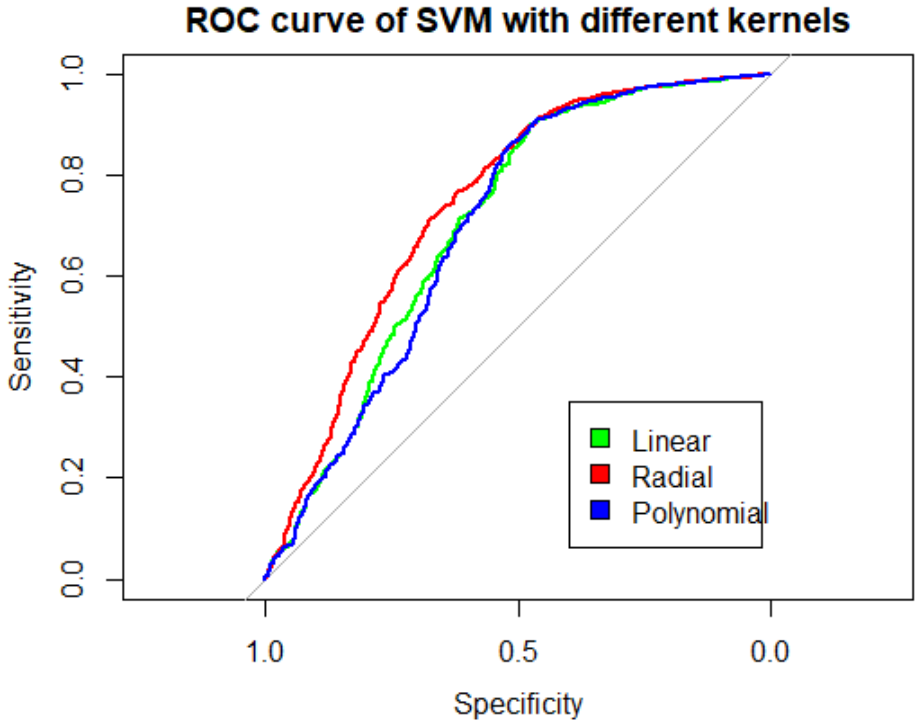


Figure 5.8: ROC curves of the SVMs with different kernels.

Table 5.6: Performance of the SVM credit scoring classifier.

Kernel	Linear	Radial	Polynomial
AUC of model	0.7041	0.7420	0.6973

Repeated 7-fold cross validation was used when training the support vector machine classifier in order to choose the optimal model. The output can be found in appendix D section D.6 on page 149 and is summarised in figure 5.9 on the next page. The cross validation results show that the optimal model has a cost value of 0.75 and a gamma value of 0.05. The

cost parameter defines the weight of how much samples inside the margin contribute to the overall error (Izenman, 2008, pg 384). Varying the value of C determines how hard or soft the large margin classification will be (Izenman, 2008, pg 384). The gamma parameter is part of the kernel function and defines how far the influence of a single training example reaches (Izenman, 2008, pg 384). The performance metrics of this model are available in table 5.7.

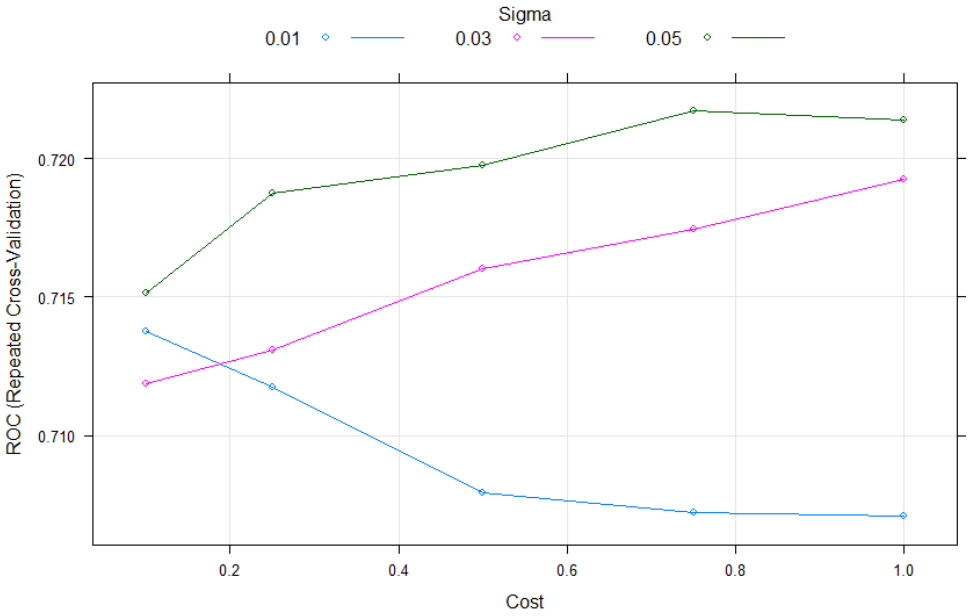


Figure 5.9: Cross validation ROC results for the SVM classifier.

Table 5.7: Performance of the SVM credit scoring classifier.

Test accuracy	AUC	Sensitivity	Specificity
0.8193	0.7424	0.9580	0.3303

5.3 Dimension Reduction

Dimension reduction was discussed in chapter 2 as a technique that can be utilised to improve classifier performance and/or interpretability. Data sets used to build credit scoring systems are often large and as a result may have a significant amount of correlated variables (Giannouli & Kountzakis, 2018). This can negatively affect the training process and result in an inefficient classifier. To address this issue dimension reduction has been used for feature selection on data sets prior to model building. The credit data set has 23 feature variables which can be considered as a high number of variables. To check if reducing the feature variable space would improve classification both the neural network and the support vector machine classifiers were rebuilt using a reduced feature space. PCA and LDA were used to reduce the feature space. The aim of this investigation was to compare the results of the

classifiers trained using the reduced feature space to the classifiers trained using the original feature space.

Hamdy & Hussein (2016) conducted an investigation where they used an NN as the classifier. They compared the accuracy, training time and the AUC of six NN classifiers. The first NN was built using the original feature space and the following five NN where built using a reduced feature space after PCA was conducted, varying the number of principal components used for each NN. They found that the NN built using the original data set performed better than the other NNs at a cost of a high training time. They found that the more principal components used to build the NN the better the performance of the NN and the closer the performance of these classifiers were to the original NN classifier. The most computationally expensive classifier was the NN built using the least number of principal components as it had the highest training time. This is because the classifier was not able to generalise quickly with the few number of feature variables. They found that, in general, all the classifiers were easier to train compared to the first NN built using the original data set and the more principal components included in the model the larger the training time. They concluded that PCA has the ability to generalise well to the credit scoring problem and its performance reduces training time.

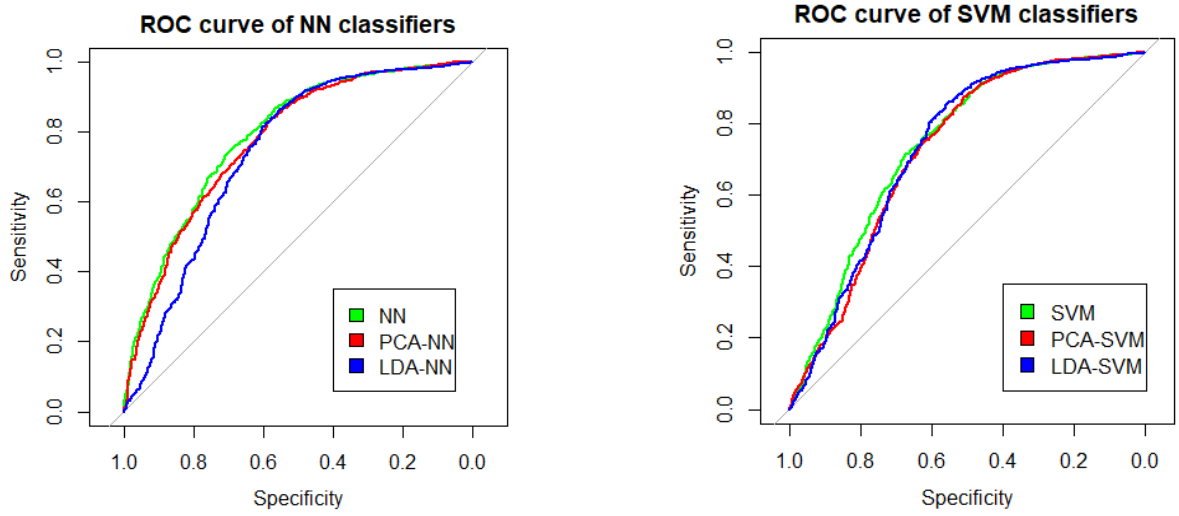
Çizer et al. (2017) conducted an investigation where they compared the classification performance between a SVM on the original feature space and a SVM on the reduced feature space using PCA as the dimension reduction technique. They found that the SVM built using the principal components resulted in a 73% accuracy rate which was 3.33% higher than the accuracy obtained from the SVM built using the original data set. They concluded that dimension reduction can result in increased classifier accuracy rate.

Marshall et al. (2010) investigated whether a bootstrap variable selection technique can result in better classifier performance when used on a large data set from a major United Kingdom retail bank. They discovered that the classification ability of a bootstrap feature selection procedure was better than the classification ability of a classifier constructed using the usual single-stepwise procedure. They concluded that bootstrap feature selection is useful in maximising the classification ability of the model.

Han et al. (2013) investigated the benefits of using orthogonal dimension reduction (ODR) in classification and compared the results to two common techniques, namely PCA and hybridizing logistic regression (HLR). They used cross validation, a paired-t test and ROC graphs to compare the performance of the dimension reduction techniques. They found that although no classifier can be completely superior to the other classifiers, the SVM classifier built on the feature space reduced by ODR improved prediction accuracy. They also found that in their case PCA was not a good choice for dimension reduction for the SVM. They concluded that ODR is a useful technique to address high dimension and has an impressive effect on SVM for credit scoring.

In this study PCA and LDA dimension reduction techniques resulted in a slight difference

in classifier performance between the classifiers built with the reduced feature space and the classifiers built with the original feature space with the latter performing slightly better. Denote the NN and SVM built with the reduced feature space using PCA as PCA-NN and PCA-SVM. Denote the classifiers built with a reduced feature space using LDA as LDA-NN and LDA-SVM. Figure 5.10a shows that the NN performs slightly better than the PCA-NN where the LDA-NN is the worst performing classifier. The PCA-NN and the LDA-NN classifiers were built using seven nodes and a weight decay of 0.3. This is similar to the result obtained by Hamdy & Hussein (2016). The PCA-NN classifier was quicker to train than the NN classifier. The SVM classifiers yielded similar results (figure 5.10b). The SVM classifier performed better than the PCA-SVM and LDA-SVM classifiers. The difference between the NN and SVM classifiers is in the accuracy of the classifiers built using the dimensionally reduced data sets. In the case of the SVM classifiers, the LDA-SVM performed better than the PCA-SVM.



(a) PCA LDA NN ROC.

(b) PCA LDA SVM ROC.

Figure 5.10: ROC curves for the NN and SVM classifiers.

Table 5.8 shows the accuracy and the AUC of these classifiers. The accuracy of these classifiers are very similar, the AUC of the classifiers are different. The AUC of these PCA-NN classifier is 3.47% higher than that of the LDA-NN classifier. The AUC of the LDA-SVM classifier is 0.68% higher than that of the PCA-SVM classifier. PCA as a dimension reduction technique outperforms LDA when neural networks are used as the classifier whereas LDA outperforms PCA when support vector machines are used as the classifier. The expectation was for the classifiers to perform better after applying the dimension reduction techniques but this is not the case.

Table 5.8: Performance results of the SVM and NN classifiers after dimension reduction.

Principal Component Analysis			Linear Discriminant Analysis		
Model	Accuracy	AUC	Model	Accuracy	AUC
NN	0.8198	0.7754	NN	0.8141	0.7407
SVM	0.8193	0.7253	SVM	0.8129	0.7321

The classifiers built after applying PCA utilised fifteen (15) principal components during model training. The results of the PCA, available in tables D.6, D.7 and D.8 in appendix D.4 on page 145, are quite difficult to interpret. The biplot of the principal components provided in figure 5.11 is also difficult to interpret. The tables indicate that the first principal component is mostly made up of variables X_{12} to X_{17} and the second principal component is mostly made up of variables X_6 to X_{11}. However it is very difficult to assess the results unless a credit data expert is consulted.

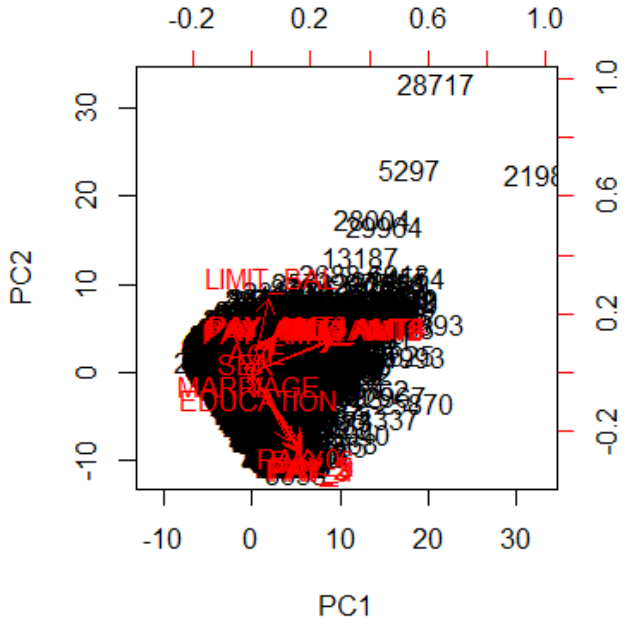
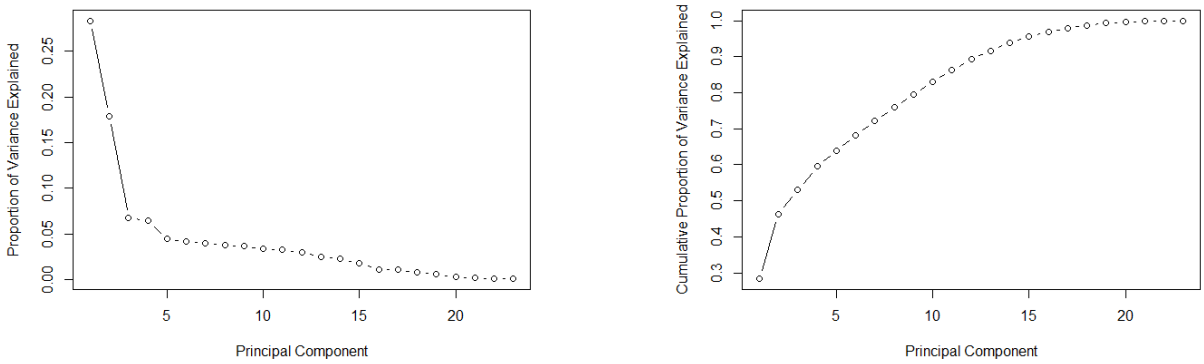


Figure 5.11: The biplot of the PCA for the credit data set.

Figures 5.12a and 5.12b show the proportion of variance explained by different numbers of principal components. Figure 5.12b converges at fifteen (15) hence fifteen (15) principal components were used to build the classifiers above. Fifteen variables is very similar to the original number of variables, namely twenty-three, used when training the original models as per sections 5.1 and 5.2. Figure 5.12a shows that the diagram elbows at three (3), five (5) and sixteen (16) principal components.

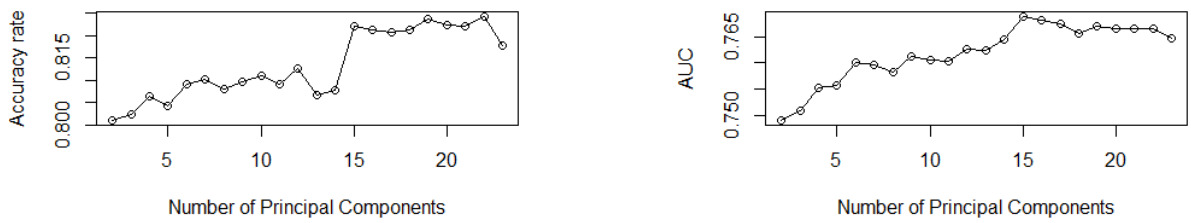


(a) Scree plot.

(b) Cumulative scree plot.

Figure 5.12: PCA scree plots for the credit data.

An investigation to determine the number of principal components which maximises the accuracy and AUC of these models was conducted. Figures 5.13a and 5.13b show how the accuracy and the AUC of PCA-NN classifiers built using a different number of principal components vary. Using twenty-two (22) principal components to build the classifier results in the highest accuracy and using fifteen (15) principal components to build the classifier resulted in the highest AUC. However the difference in the performance measures when varying the number of principal components used when training is relatively small.

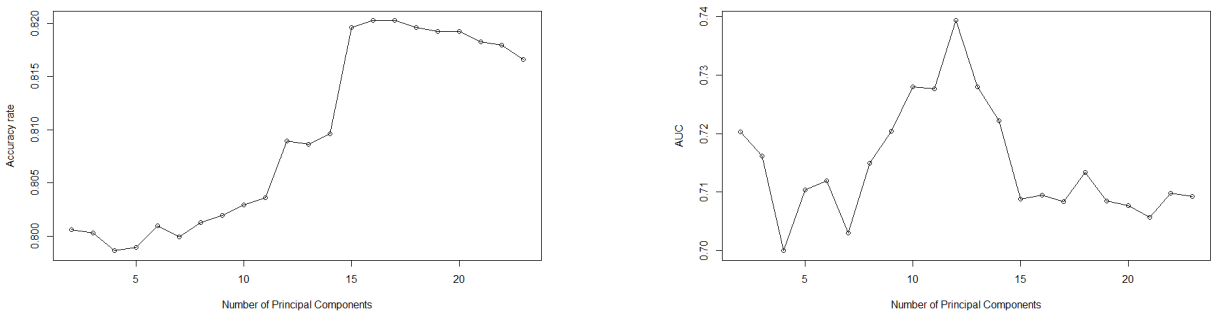


(a) Accuracies of NN's built using various numbers of principal components.

(b) AUC of NN's built using various numbers of principal components.

Figure 5.13: PCA-NN accuracy and AUC using different numbers of principal components.

Similar results were found for SVMs constructed using a radial kernel and various numbers of principal components, figures 5.14a and 5.14b. Using sixteen (16) and seventeen (17) principal components results in the highest accuracy and using twelve (12) principal components results in the highest AUC. However the overall difference is also minimal. This confirms that in the context of this analysis, dimension reduction does not significantly reduce the model accuracy. An expert might be able to interpret the principal components and hence these simpler models, with similar accuracy, may be easier to interpret.



(a) Accuracies of SVM's built using various numbers of principal components.

(b) AUC of SVM's built using various numbers of principal components.

Figure 5.14: PCA-SVM accuracy and AUC using different numbers of principal components.

5.4 Comparison

Yeh & Lien (2009) used six data mining techniques to estimate the probability of credit default of customers in a bank located in Taiwan. They believed that estimating probability of default is more meaningful than classifying the customers into risky and not risky clients. Since the real probability of default is unknown they proposed the model "Sorting Smoothing Method" to deduce the real probability of default and offered solutions to two questions:

- Is there a difference between classification accuracy across the data mining techniques?
- Could the estimated probability of default produced from the data mining techniques represent the real probability of default?

In this study two data mining techniques were used to construct credit scoring systems and their performance is compared to identify the better performing technique. Similar to this study, Yeh & Lien (2009) noted that using accuracy/error rate to compare classifier performance was not ideal since most of the instances in the data set are non-risky customers. As an alternative they used area ratio of lift charts. In this study the AUC was used as the alternative performance measure. One of the data mining techniques used in their study is a neural network model which was one of the two classifiers used in this study. They found that out of all the data mining techniques the neural network classifier performed the best as its lift chart had the highest area ratio and it had the lowest error rate. The same result was obtained in this study as the neural network classifier outperformed the support vector classifier. The neural network classifier had a lower error rate and a higher AUC value compared to the support vector machine classifier. An important fact to note is that similarly to this study, they found that the difference in classification accuracy across classifiers was minimal.

Chapter 6

Conclusion

Credit scoring systems help financial institutions predict potential credit default customers and hence assist in making well informed decisions regarding awarding credit. In this study two classification techniques were used to develop credit scoring systems, namely neural networks and support vector machines. Principal component analysis and linear discriminant analysis were used to reduce the dimension of the credit data set. The main aim of this study was to compare the performance of the resulting classifiers and determine if reducing the dimension of the data set would improve the performance of the classifiers.

6.1 Limitations of this Study

All models contructed in this study were built on desktop intel core i5 PC's. The researcher did not have access to cloud computing infrastructure. At the onset of this study it was assumed that the modest size of the credit data set would not require these resources.

6.2 Neural Networks

The neural network classifier achieved the highest accuracy and AUC value. Repeated 7-fold cross validation was used to validate the classifier. When the dimension of the data set were reduced using PCA and LDA the neural network classifier trained using the feature space reduced by the PCA method performed better than the neural network classifier trained using the feature space reduced by the LDA method. Both classifiers resulting from the reduced feature space performed as well as the neural network trained using the original data set but they failed to perform better. The advantage of using dimension reduction techniques was a less computationally expensive neural network classifier. The NN constructed using the optimal number of principal components failed to perform better than the neural network built using the original data set. This lead to the conclusion that in the context of this

problem, the dimension reduction techniques are not effective in improving the neural network classifier performance.

6.3 Support Vector Machines

The support vector machine classifier performed well but not better than the neural network classifier. Repeated 7-fold cross validation was used to validate the classifier. After the data set was reduced using PCA and LDA the support vector machine classifier built using the LDA reduced feature space performed better than the support vector machine classifier built using the PCA reduced feature space. Similarly to the neural network classifiers, the classifiers trained using the reduced feature space did not perform better than the support vector machine classifier trained on the original data set but did result in less computationally expensive classifiers. Twelve principal components resulted in the optimal reduced feature space support vector machine classifier. This model failed to perform better than the support vector machine trained on the original data set. This lead to the same conclusion as that of neural network classifiers, dimension reduction techniques are not effective in improving support vector machine classifier performance in the context of this analysis.

6.4 Challenges in this Study

Training the support vector machine classifiers was computationally expensive. The polynomial kernel support vector machine was the most computationally expensive classifier out of all of the classifiers. The models took between two to four days to train with most initially producing errors. The SVM classifier took a long time to train due to the relatively large number of observations in the train data set paired with the number of iterations and the tune grid used to train the classifier. The tune grid is a set of possible values for the parameters of a classifier from which the values that optimise model performance are chosen. There are approximately twenty-seven thousand observations in the training data set using the 90/10 train/test split to partition the original data. To check if the three classifiers with the different kernels would eventually finish training a smaller subset of the data set, about fifteen percent (15%), was used to train the classifiers. The outcome proved that the models were in fact training and the analyst needed some patience. The results of the classifiers trained using the smaller subset of the data were used to reduce the tune grid as a means to lessen the number of iterations used to train the classifiers on the full train data set. As a result all three SVM models eventually finished training hence it was possible to identify the best performing model, namely the support vector machine with the radial kernel. For the support vector machine with a radial kernel and trained with repeated 7-fold cross validation, the full train data set and original tune grid were used to build the model. This took

close to a week to train. This allowed for the optimal support vector machine classifier to be identified with confidence. Patience and investigation were key factors in completing this part of the study.

6.5 Future Research

There are several avenues that can be considered for future research. The trained neural network classifiers had one hidden layer. Increasing the number of hidden layers may result in a better performing neural network classifier. Alternative dimension reduction techniques, for example ODR and HLR, may be used to reduce the feature space of the data set. This may result in dimension reduction being an effective tool in improving model performance. Most feature variables in the credit data set have a number of outliers. Finding a more suitable method to deal with the outliers may result in a better performing classifier in the context of this application. The SVM and NN classifiers constructed in this thesis were not easilly interpretable. The resulting classifiers are complex black-boxes and are not effective in reducing the time taken when completing documentation by customers applying for credit. The reduced set of feature variables used to construct the classifiers in this study did not identify variables in the original that could be ommitted. The credit data set used in this study has a limitted number of feature variables to construct a customer profile. The data set unfortunately does not include detailed information about the applicant's credit history relative to their remunation, their cash or saving balance history, investments and assets. In addition if the applicant is married the data for the couple are combined in one data entry and not reflected over different variables. As a result complex customer credit profiles or credit risk profiles (Lanzarini et al., 2017) were not considered in this study.

References

Abe, S. (2005). *Support Vector Machines for Pattern Classification*. Springer, First edition.

Angelini, E., di Tollo, G., & Roli, A. (2008). A neural network approach for credit risk evaluation. *The Quarterly Review of Economics and Finance*, 48(4), 733–755.

Basheer, I. A. & Hajmeer, M. (2000). Artificial neural networks: Fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1), 3–31.

Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM Sigkdd Explorations Newsletter*, 6(1), 20–29.

Chawla, N. V., Japkowicz, N., & Kotcz, A. (2004). Special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter*, 6(1), 1–6.

Çizer, E. B., Ayça, A., & Topuz, V. (2017). Credit repayment analysis using support vector machine and principal component analysis. *International Journal of Social and Economic Sciences (IJSES) E-ISSN: 2667-4904*, 7(2), 22–25.

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4), 547–553.

Dongare, A., Kharde, R., & Kachare, A. D. (2012). Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1), 189–194.

Elhabian, S. & Farag, A. A. (2009). *Linear Discriminant Analysis, Lecture Notes*. University of Louisville, CVIP Lab.

Ennett, C. M., Frize, M., & Walker, C. R. (2001). *Influence of Missing Values on Artificial Neural Network Performance*. Medinfo, IOS Press.

Everitt, B. S. & Hothorn, T. (2006). *A Handbook of Statistical Analyses using R*. Chapman & Hall/CRC.

Fausett, L. (1994). *Fundamentals of Neural Networks*. Prentice-Hall International, First edition.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), 861–874.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2), 179–188.

Franc, V. & Hlaváč, V. (2003). An iterative algorithm learning the maximal margin classifier. *Pattern Recognition*, 36(9), 1985–1996.

Franceschi, S., Talamini, R., Barra, S., Barón, A. E., Negri, E., Bidoli, E., Serraino, D., & La Vecchia, C. (1990). Smoking and drinking in relation to cancers of the oral cavity, pharynx, larynx, and esophagus in Northern Italy. *Cancer Research*, 50(20), 6502–6507.

Freund, Y. & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3), 277–296.

Gentile, C. (2001). A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2(Dec), 213–242.

Giannouli, P. & Kountzakis, C. E. (2018). *The use of PCA in reduction of credit scoring modeling variables: Evidence from Greek banking system*. `http://www.preprints.org`, accessed July 2019.

Golden, R. M. (1986). The "Brain-State-in-a-Box" neural model is a gradient descent algorithm. *Journal of Mathematical Psychology*, 30(1), 73–80.

Hair, J. F., Anderson, R. E., Tatham, R. L., & Black, W. C. (1998). *Multivariate Data Analysis*. Prentice-Hall International, Fifth edition.

Hamdy, A. & Hussein, W. B. (2016). Credit risk assessment model based using principal component analysis and artificial neural network. In *MATEC Web of Conferences*, volume 76(02039): EDP Sciences.

Han, L., Han, L., & Zhao, H. (2013). Orthogonal support vector machine for credit scoring. *Engineering Applications of Artificial Intelligence*, 26(2), 848–862.

Hanif, M. (2015). Mathematics for human nervous system. *International Journal of Mathematics Trends and Technology*, 27(1), 12–18.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning*. Springer.

Haykin, S. (1994). *Neural Networks*. Macmillan, First edition.

Islam, S. R., Eberle, W., & Ghafoor, S. K. (2018). Credit default mining using combined machine learning and heuristic approach. *arXiv preprint arXiv:1807.01176*.

Izenman, A. J. (2008). *Modern Multivariate Statistical Techniques*. Springer, New York.

Jäger, G. (2005). *Neural Networks, Lecture Notes*. Rhodes University.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer.

Jayalakshmi, T. & Santhakumaran, A. (2011). Statistical normalization and back propagation for classification. *International Journal of Computer Theory and Engineering*, 3(1), 1793–8201.

Johnson, R. A. & Wichern, D. W. (1992). *Applied Multivariate Statistical Analysis*. Prentice-Hall International, Third edition.

Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer-Verlag, Second edition.

Karatzoglou, A., Meyer, D., & Hornik, K. (2006). Support vector machines in R. *Journal of Statistical Software*, 15(9), 1–28.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Ijcai*, 14(2), 1137–1145.

Kun, Z., Ying-jie, T., & Nai-yang, D. (2006). Unsupervised and semi-supervised two-class support vector machines. In *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)* (pp. 813–817).: IEEE.

Lanzarini, L. C., Villa Monte, A., Bariviera, A. F., & Jimbo Santana, P. (2017). Simplifying credit scoring rules using lvq+ pso. *Kybernetes*, 46(1), 8–16.

Laskov, P., Düssel, P., Schäfer, C., & Rieck, K. (2005). Learning intrusion detection: supervised or unsupervised? In *International Conference on Image Analysis and Processing* (pp. 50–57).: Springer.

Lauret, P., Fock, E., Randrianarivony, R. N., & Manicom-Ramsamy, J.-F. (2008). Bayesian neural network approach to short time load forecasting. *Energy Conversion and Management*, 49(5), 1156–1166.

Li, C. & Wang, B. (2014). *Fisher Linear Discriminant Analysis*. `http://www.ccs.neu.edu/home/vip/teach/MLcourse/5_features_dimensions/lecture_notes/LDA/LDA.pdf`, accessed May 2018.

Ling, C. X., Huang, J., Zhang, H., et al. (2003). AUC: A statistically consistent and more discriminating measure than accuracy. *Ijcai*, 3, 519–524.

Mammone, A., Turchi, M., & Cristianini, N. (2009a). Support vector machines. *Wires Computational Statistics*, 1, 283–289.

Mammone, A., Turchi, M., & Cristianini, N. (2009b). Support vector machines. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(3), 283–289.

Marshall, A., Tang, L., & Milne, A. (2010). Variable reduction, sample selection bias and bank retail credit scoring. *Journal of Empirical Finance*, 17(3), 501–512.

Martínez, A. M. & Kak, A. C. (2001). PCA versus LDA. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2), 228–233.

May, R. J., Maier, H. R., & Dandy, G. C. (2010). Data splitting for artificial neural networks using som-based stratified sampling. *Neural Networks*, 23(2), 283–294.

Mayer, D. & Butler, D. (1993). Statistical validation. *Ecological Modelling*, 68(1-2), 21–32.

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4), 115–133.

Mester, L. J. (1997). What is the point of credit scoring? *Business Review*, 3(Sep/Oct), 3–16.

Müller, B., Reinhardt, J., & Strickland, M. T. (1995). *Neural Networks*. Springer, Second edition.

Panchal, F. S. & Panchal, M. (2014). Review on methods of selecting number of hidden nodes in artificial neural network. *International Journal of Computer Science and Mobile Computing*, 3(11), 455–464.

Polat, K. (2012). Classification of Parkinson's disease using feature weighting method on the basis of fuzzy C-means clustering. *International Journal of Systems Science*, 43(4), 597–609.

Radloff, S. (2014). *Mathematical Statistics 301: The General Linear Model, Lecture Notes*. Rhodes University.

Ramón y Cajál, S. (1911). Histogénèse du cervelet. In Haykin (1994).

Riedmiller, M. (1994). Advanced supervised learning in multi-layer perceptrons-from back-propagation to adaptive learning algorithms. *Computer Standards & Interfaces*, 16(3), 265–278.

Stephen, I. (1990). Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 50(2), 179.

Tenenbaum, J. B., De Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500), 2319–2323.

Tharwat, A. (2016). Principal component analysis - a tutorial. *International Journal of Applied Pattern Recognition*, 3, 197.

Ungar, L., Powell, B., & Kamens, S. (1990). Adaptive networks for fault diagnosis and process control. *Computers & Chemical Engineering*, 14(4-5), 561–572.

Van Der Maaten, L., Postma, E., & Van den Herik, J. (2009). *Dimensionality Reduction: A Comparative Review*. Tilburg University, Tilburg, Netherlands. `https://lvdmaaten.github.io/publications/papers/TR_Dimensionality_Reduction_Review_2009.pdf`, accessed May 2018.

Veelenturf, L. P. J. (1995). *Analysis and Application of Artificial Neural Networks*. Prentice-Hall International, First edition.

Verleysen, M. & François, D. (2005). The curse of dimensionality in data mining and time series prediction. In *International Work-Conference on Artificial Neural Networks* (pp. 758–770).: Springer.

Yeh, I.-C. & Lien, C.-h. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2), 2473–2480.

Zaki, M. J., Meira Jr, W., & Meira, W. (2014). *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press.

Zhang, X. (2011). : (pp. 941–946). Springer.

Appendix A

The Wine Data Set

A.1 First Ten Observations: The Wine Data

The output of the first ten observation of the wine data set in chapter 2 is as follows

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids
1	one	14.23	1.71	2.43	15.6	127	2.80
2	one	13.20	1.78	2.14	11.2	100	2.65
3	one	13.16	2.36	2.67	18.6	101	2.80
4	one	14.37	1.95	2.50	16.8	113	3.85
5	one	13.24	2.59	2.87	21.0	118	2.80
6	one	14.20	1.76	2.45	15.2	112	3.27
7	one	14.39	1.87	2.45	14.6	96	2.50
8	one	14.06	2.15	2.61	17.6	121	2.60
9	one	14.83	1.64	2.17	14.0	97	2.80
10	one	13.86	1.35	2.27	16.0	98	2.98

	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline
1	0.28	2.29	5.64	1.04	3.92	1065
2	0.26	1.28	4.38	1.05	3.40	1050
3	0.30	2.81	5.68	1.03	3.17	1185
4	0.24	2.18	7.80	0.86	3.45	1480
5	0.39	1.82	4.32	1.04	2.93	735
6	0.34	1.97	6.75	1.05	2.85	1450
7	0.30	1.98	5.25	1.02	3.58	1290
8	0.31	1.25	5.05	1.06	3.58	1295
9	0.29	1.98	5.20	1.08	2.85	1045
10	0.22	1.85	7.22	1.01	3.55	1045

Table A.1: First ten observation of the wine data set.

A.2 Summary of these Data

The output of the summary for the wine data set in chapter 2 is as follows

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids
Min	11.03	0.740	1.360	10.60	70.00	0.980	0.340
Q_1	12.36	1.603	2.210	17.20	88.00	1.742	1.205
$\tilde{x}$	13.05	1.865	2.360	19.50	98.00	2.355	2.135
$\overline{x}$	13.00	2.336	2.367	19.49	99.74	2.295	2.029
Q_3	13.68	3.083	2.558	21.50	107.00	2.800	2.875
Max	14.83	5.800	3.230	30.00	162.00	3.880	5.080

	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline
Min	0.1300	0.410	1.280	0.4800	1.270	278.0
Q_1	0.2700	1.250	3.220	0.7825	1.938	500.5
$\tilde{x}$	0.3400	1.555	4.690	0.9650	2.780	673.5
$\overline{x}$	0.3619	1.591	5.058	0.9574	2.612	746.9
Q_3	0.4375	1.950	6.200	1.1200	3.170	985.0
Max	0.6600	3.580	13.000	1.7100	4.000	1680.0

Table A.2: Summary statistics of the variables in the wine data set.

A.3 Variance-Covariance Matrix

The output of the variance-covariance matrix of the wine data in chapter 2 is as follows

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids
Alcohol	1.0000	0.0944	0.2115	−0.3102	0.2708	0.2891	0.2368
Malic	0.0944	1.0000	0.1640	0.2885	−0.0546	−0.3352	−0.4110
Ash	0.2115	0.1640	1.0000	0.4434	0.2866	0.1290	0.1151
Alcalinity	−0.3102	0.2885	0.4434	1.0000	−0.0833	−0.3211	−0.3514
Magnesium	0.2708	−0.0546	0.2866	−0.0833	1.0000	0.2144	0.1958
Phenols	0.2891	−0.3352	0.1290	−0.3211	0.2144	1.0000	0.8646
Flavanoids	0.2368	−0.4110	0.1151	−0.3514	0.1958	0.8646	1.0000
Nonflavanoids	−0.1559	0.2930	0.1862	0.3619	−0.2563	−0.4499	−0.5379
Proanthocyanins	0.1367	−0.2207	0.0097	−0.1973	0.2364	0.6124	0.6527
Color	0.5464	0.2490	0.2589	0.0187	0.2000	−0.0551	−0.1724
Hue	−0.0717	−0.5613	−0.0747	−0.2740	0.0554	0.4337	0.5435
Dilution	0.0723	−0.3687	0.0039	−0.2768	0.0660	0.6999	0.7872
Proline	0.6437	−0.1920	0.2236	−0.4406	0.3934	0.4981	0.4942

	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline
Alcohol	−0.1559	0.1367	0.5464	−0.0717	0.0723	0.6437
Malic	0.2930	−0.2207	0.2490	−0.5613	−0.3687	−0.1920
Ash	0.1862	0.0097	0.2589	−0.0747	0.0039	0.2236
Alcalinity	0.3619	−0.1973	0.0187	−0.2740	−0.2768	−0.4406
Magnesium	−0.2563	0.2364	0.2000	0.0554	0.0660	0.3934
Phenols	−0.4499	0.6124	−0.0551	0.4337	0.6999	0.4981
Flavanoids	−0.5379	0.6527	−0.1724	0.5435	0.7872	0.4942
Nonflavanoids	1.0000	−0.3658	0.1391	−0.2626	−0.5033	−0.3114
Proanthocyanins	−0.3658	1.0000	−0.0252	0.2955	0.5191	0.3304
Color	0.1391	−0.0252	1.0000	−0.5218	−0.4288	0.3161
Hue	−0.2626	0.2955	−0.5218	1.0000	0.5655	0.2362
Dilution	−0.5033	0.5191	−0.4288	0.5655	1.0000	0.3128
Proline	−0.3114	0.3304	0.3161	0.2362	0.3128	1.0000

Table A.3: Variance-covariance matrix of the wine data set.

A.4 Principal Component Analysis Results

The results of the PCA of the wine data in chapter 2 is as follows

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
Alcohol	−0.1239	0.5027	−0.1579	0.0285	−0.2219	0.2040	−0.1024
Malic	0.2526	0.1552	0.2069	0.4820	0.2708	0.5683	0.3738
Ash	0.0286	0.3538	0.5807	−0.2120	−0.2070	0.0856	−0.1609
Alcalinity	0.2432	−0.0003	0.6154	−0.1058	0.0754	−0.1468	−0.2798
Magnesium	−0.1378	0.3058	0.0509	−0.5126	0.6405	−0.0816	0.3325
Phenols	−0.3828	0.0727	0.1709	0.2242	−0.0700	−0.0838	−0.0635
Flavanoids	−0.4271	0.0002	0.1370	0.1709	−0.0674	0.0345	−0.0758
Nonflavanoids	0.3171	0.0590	0.1280	−0.0335	−0.5232	−0.1618	0.5966
Proanthocyanins	−0.2994	0.0766	0.1987	0.4221	0.1455	−0.5522	0.3426
Colour	0.1182	0.5117	−0.2025	0.1563	−0.0688	−0.3410	−0.1859
Hue	−0.3100	−0.2425	0.0849	−0.3645	−0.2832	0.0436	0.3080
Dilution	−0.3744	−0.1471	0.2037	0.0911	−0.0228	0.2952	−0.0603
Proline	−0.2692	0.3840	−0.1421	−0.1537	−0.1695	0.2350	0.1374

	PC8	PC9	PC10	PC11	PC12	PC13
Alcohol	0.4038	−0.5090	0.2297	0.1891	0.2736	0.1388
Malic	0.0961	−0.0360	−0.2569	0.0209	−0.1414	−0.0632
Ash	−0.1649	0.3049	0.0790	0.4922	−0.2045	0.0725
Alcalinity	0.3254	−0.1550	−0.1369	−0.4839	0.2462	−0.0442
Magnesium	−0.1720	−0.2175	0.0949	−0.0270	−0.0038	−0.0889
Phenols	−0.5182	−0.3484	−0.4206	−0.0917	0.0730	0.4150
Flavanoids	−0.1155	−0.0425	−0.0572	0.0971	0.2461	−0.8184
Nonflavanoids	−0.2844	−0.1842	0.2223	−0.1769	0.1079	−0.1244
Proanthocyanins	0.3415	0.2257	0.1683	0.1243	0.0550	0.1703
Color	0.0073	−0.0647	−0.1615	−0.2409	−0.5943	−0.2480
Hue	0.4271	−0.1607	−0.4258	0.0730	−0.3638	−0.0265
Dilution	−0.0394	−0.0852	0.5973	−0.3929	−0.4158	0.0650
Proline	0.0404	0.5744	−0.1584	−0.4521	0.2522	0.0936

Table A.4: PCA results of the wine data set.

A.5 R Code for the Wine Data Example

The R script for the wine data set example in chapter 2 is as follows

```
rm(list=ls())
library(caret)

# read in data
mydata <- read.csv("wine.csv", sep = ",", header = FALSE)
str(mydata)

# is there any missing data?
apply(mydata,2,function(x) sum(is.na(x)))
```

```
# changing variable names
colnames(mydata) <- c('Type', 'Alcohol', 'Malic', 'Ash', 'Alcalinity', '
    Magnesium', 'Phenols', 'Flavanoids', 'Nonflavanoids', 'Proanthocyanins', '
    Colour', 'Hue', 'Dilution', 'Proline')

# changing class names
indexes <- which(mydata$Type == 1)
mydata$Type[indexes] <- 'one'
indexes <- which(mydata$Type == 2)
mydata$Type[indexes] <- 'two'
indexes <- which(mydata$Type == 3)
mydata$Type[indexes] <- 'three'

# changing response variable to factor
mydata$Type <- as.factor(mydata$Type)

# distribution of classes
cbind(freq = table(mydata$Type), percentage = prop.table(table(mydata$Type))*
    100)

# have a look at the first ten observation
head(mydata, n=10)
# output to latex for thesis
library(Hmisc)
latex(head(mydata, n=10), file="wine10obs.tex")

# simple summary statistics
summary(mydata)
# create a latex table
latex(summary(mydata), file="winesummarystats.tex")

# variance covariance matrix
var.cor <- cor(mydata[,2:14])
# create a latex table
latex(round(var.cor,4), file="winecorrelations.tex")

# diagram of covariance
ggcorr(mydata[,2:14])

# drawing box plot of each feature variables
par(mfrow=c(2,2))
boxplot(mydata[,2], main=paste("Boxplot of ", names(mydata)[2]), horizontal=
    TRUE)
boxplot(mydata[,3], main=paste("Boxplot of ", names(mydata)[3]), horizontal=
    TRUE)
boxplot(mydata[,4], main=paste("Boxplot of ", names(mydata)[4]), horizontal=
    TRUE)
```

```
boxplot(mydata[,5], main=paste("Boxplot of ", names(mydata)[5]), horizontal=
    TRUE)
dev.print(device=postscript, file="Boxplots1.eps")
dev.off()

par(mfrow=c(2,2))
boxplot(mydata[,6], main=paste("Boxplot of ", names(mydata)[6]), horizontal=
    TRUE)
boxplot(mydata[,7], main=paste("Boxplot of ", names(mydata)[7]), horizontal=
    TRUE)
boxplot(mydata[,8], main=paste("Boxplot of ", names(mydata)[8]), horizontal=
    TRUE)
boxplot(mydata[,9], main=paste("Boxplot of ", names(mydata)[9]), horizontal=
    TRUE)
dev.print(device=postscript, file="Boxplots2.eps")
dev.off()

par(mfrow=c(2,2))
boxplot(mydata[,10], main=paste("Boxplot of ", names(mydata)[10]), horizontal=
    TRUE)
boxplot(mydata[,11], main=paste("Boxplot of ", names(mydata)[11]), horizontal=
    TRUE)
boxplot(mydata[,12], main=paste("Boxplot of ", names(mydata)[12]), horizontal=
    TRUE)
boxplot(mydata[,13], main=paste("Boxplot of ", names(mydata)[13]), horizontal=
    TRUE)
dev.print(device=postscript, file="Boxplots3.eps")
dev.off()

par(mfrow=c(1,1))
boxplot(mydata[,14], main=paste("Boxplot of ", names(mydata)[14]), horizontal=
    TRUE)
dev.print(device=postscript, file="Boxplots4.eps")
dev.off()

# drawing boxplot of all feature variables
boxplot(mydata[,2:13])
#
plot(mydata[,2:13], col=c("red","blue","green")[mydata$Type])
dev.print(device=postscript, file="WineScatterplotClass.eps")
# checking for outliers
outvals <- boxplot(mydata[,2:8], plot = FALSE)$out

# checking distribution of data set using selected feature variables
with(mydata, qplot(Alcohol, Malic, colour=Type, cex=2))

#########################################
# Use a classification tree to classify the wine type
```

```
# Divide data set into train and test set: 70/30
# Fit to the original data
#####################################

# train/test split
# indexes <- createDataPartition(mydata$Type, times = 1, p = 0.7, list = FALSE
    )
# save(indexes, file="E://Project//Rscript//indexesWine.Rdata")

load("indexesWine.Rdata")

train <- mydata[indexes,]
test <- mydata[-indexes,]

# Fit the tree/build the classifier
library(rpart)
treeModel <- rpart(Type ~ ., data=train)

# output: Interpret the model
treeModel
summary(treeModel)

# plot tree
par(xpd=TRUE)
plot(treeModel, compress=TRUE)
text(treeModel, use.n=TRUE)
dev.print(device=postscript, file="ClassificationTree_originaldata.eps")

# confusion matrix and accuracy of training data and test data:
confusionMatrix(predict(treeModel, newdata=train[,-1], type="class"), train
    [,1])
confusionMatrix(predict(treeModel, newdata=test[,-1], type="class"), test[,1])

#######################################
# Use a classification tree to classify the wine type
# Divide data set into train and test set: 70/30
# Fit to the reduced data: PCA
#
# Conduct PCA
# Fit model
#######################################

prin_comp <- prcomp(mydata[,-1], scale. = T)

# Interpretation
prin_comp
latex(round(prin_comp$rotation,4), file="PrinCompScoresWine.latex")
prin_comp$rotation
```

```
# Interpretation: Not overly clear unless you are a wine expert or chemist?
biplot(prin_comp, scale = 0)

# compute standard deviation of each principal component
std_dev <- prin_comp$sdev

# compute variance
pr_var <- std_dev^2

# check variance of first 10 components
pr_var[1:10]

# proportion of variance explained
prop_varex <- pr_var/sum(pr_var)
prop_varex[1:10]
sum(prop_varex[1:8])

# scree plot
plot(prop_varex, xlab = "Principal Component", ylab = "Proportion of Variance
    Explained", type = "b")

# cumulative scree plot
plot(cumsum(prop_varex), xlab = "Principal Component", ylab = "Cumulative
    Proportion of Variance Explained", type = "b")

# add a data set with principal components
mydataPCA <- data.frame(Type = mydata$Type, prin_comp$x)

trainPCA <- mydataPCA[indexes,]
testPCA <- mydataPCA[-indexes,]

# we are interested in first 8 PCAs
trainPCA1 <- trainPCA[,1:9]

# Fit the tree/build the classifier
treeModelPCA = rpart(Type ~ ., data=trainPCA1)

# output: Interpret the model
treeModelPCA
summary(treeModelPCA)

# plot tree
par(xpd=TRUE)
plot(treeModelPCA, compress=TRUE)
text(treeModelPCA, use.n=TRUE)
dev.print(device=postscript, file="ClassificationTree_PCAdata.eps")

# confusion matrix and accuracy of training data:
```

```
confusionMatrix(predict(treeModelPCA, newdata=trainPCA1[,-1], type="class"),
    train[,1])

# confusion matrix and accuracy of test data:
# predict using PCA test data named testPCA.

testPCA1 <- testPCA[,1:9]

confusionMatrix(predict(treeModelPCA, newdata=testPCA1[,-1], type="class"),
    test[,1])

########################################
# Use a classification tree to classify the wine type
# Divide data set into train and test set: 70/30
# Fit to the reduced data: LDA
#
# Conduct lda
# Fit model
########################################

library( MASS )
linearDis = lda(Type ~ ., data = mydata )

# Interpretation
projected_data = as.matrix( mydata[, 2:14] ) %*% linearDis$scaling
plot( projected_data, col = mydata[,1], pch = 19 )

# add a training set with linear discriminants
mydataLDA <- as.matrix( mydata[, 2:14] ) %*% linearDis$scaling
mydataLDA <- as.data.frame(mydataLDA)
mydataLDA <- data.frame(Type = mydata$Type, mydataLDA)

trainLDA <- mydataLDA[indexes,]
testLDA <- mydataLDA[-indexes,]

# Fit the tree/build the classifier
treeModelLDA = rpart(Type ~ ., data=trainLDA)

# output: Interpret the model
treeModelLDA
summary(treeModelLDA)

# plot tree
par(xpd=TRUE)
plot(treeModelLDA, compress=TRUE)
text(treeModelLDA, use.n=TRUE)
dev.print(device=postscript, file="ClassificationTree_LDAdata.eps")
```

```
# confusion matrix and accuracy of training data:
confusionMatrix(predict(treeModelLDA, newdata=trainLDA[,-1], type="class"),
    train[,1])

# confusion matrix and accuracy of test data:
# predict using LDA test data named testLDA.

confusionMatrix(predict(treeModelLDA, newdata=testLDA[,-1], type="class"),
    test[,1])

#####################################
# 7-fold cross validation, repeated 10 times on the original data
# loading model package
#####################################

ctrl.1 <- trainControl(method = 'repeatedcv', number = 7, repeats = 10)

# Fit the tree/build the classifier
treeModelCVorig <- train(Type ~ ., data=mydata, method="rpart", preProcess = c
    ("center", "scale"), trControl = ctrl.1)

#####################################
# 7-fold crossvalidation, repeated 10 times on the LDA data
#####################################

# Fit the tree/build the classifier
treeModelCVPCA <- train(Type ~ ., data=mydataPCA, method="rpart", trControl =
    ctrl.1)

#####################################
# 7-fold crossvalidation, repeated 10 times on the PCA data
#####################################

# Fit the tree/build the classifier
treeModelCVLDA <- train(Type ~ ., data=mydataLDA, method="rpart", trControl =
    ctrl.1)

#####################################
#####################################
```

Listing A.1: Wine Data PCA LDA R script

A.6 R Output for these Models

The R script for the output of the classification models built for wine data set example in chapter 2 is as follows

```
##############################
# Original data: CART
##############################

Call:
  rpart(formula = Type ~ ., data = train)
n= 126

CP nsplit rel error    xerror       xstd
1 0.48684211      0 1.0000000 1.1184211 0.06919943
2 0.35526316      1 0.5131579 0.5394737 0.06919943
3 0.01315789      2 0.1578947 0.2631579 0.05397207
4 0.01000000      3 0.1447368 0.2368421 0.05168318

Variable importance
Flavanoids          Proline         Dilution          Alcohol           Colour
19               14               13               12                9
Phenols              Hue Proanthocyanins       Alcalinity        Magnesium
8                8                5                5                5

Node number 1: 126 observations,    complexity param=0.4868421
predicted class=two    expected loss=0.6031746  P(node) =1
class counts:    42    34    50
probabilities: 0.333 0.270 0.397
left son=2 (47 obs) right son=3 (79 obs)
Primary splits:
  Proline    < 727.5  to the right, improve=32.93511, (0 missing)
Alcohol    < 12.78  to the right, improve=31.30324, (0 missing)
Colour     < 3.915  to the right, improve=30.20491, (0 missing)
Dilution   < 2.115  to the right, improve=27.63468, (0 missing)
Flavanoids < 1.425  to the right, improve=26.62373, (0 missing)
Surrogate splits:
  Flavanoids < 2.31   to the right, agree=0.849, adj=0.596, (0 split)
Phenols    < 2.335  to the right, agree=0.817, adj=0.511, (0 split)
Alcohol    < 13.235 to the right, agree=0.770, adj=0.383, (0 split)
Alcalinity < 17.45  to the left,  agree=0.770, adj=0.383, (0 split)
Magnesium  < 99.5   to the right, agree=0.762, adj=0.362, (0 split)

Node number 2: 47 observations,    complexity param=0.01315789
predicted class=one    expected loss=0.1276596  P(node) =0.3730159
class counts:    41     2     4
probabilities: 0.872 0.043 0.085
left son=4 (40 obs) right son=5 (7 obs)
Primary splits:
  Flavanoids < 2.35   to the right, improve=4.287082, (0 missing)
Dilution   < 2.7    to the right, improve=4.287082, (0 missing)
Colour     < 3.75   to the right, improve=3.579939, (0 missing)
Alcohol    < 13.06  to the right, improve=3.013639, (0 missing)
```

```
Phenols    < 2.405  to the right, improve=2.437082, (0 missing)
Surrogate splits:
  Dilution < 2.64   to the right, agree=0.957, adj=0.714, (0 split)
Phenols  < 2.125  to the right, agree=0.936, adj=0.571, (0 split)
Alcohol  < 12.66  to the right, agree=0.915, adj=0.429, (0 split)
Colour   < 3.2    to the right, agree=0.894, adj=0.286, (0 split)
Hue      < 0.785  to the right, agree=0.894, adj=0.286, (0 split)

Node number 3: 79 observations,    complexity param=0.3552632
predicted class=two    expected loss=0.4177215  P(node) =0.6269841
class counts:     1    32    46
probabilities: 0.013 0.405 0.582
left son=6 (33 obs) right son=7 (46 obs)
Primary splits:
  Dilution   < 2.115  to the left,  improve=28.09031, (0 missing)
Colour     < 4.79   to the right, improve=26.52762, (0 missing)
Flavanoids < 1.425  to the left,  improve=23.96580, (0 missing)
Alcohol    < 12.745 to the right, improve=20.42492, (0 missing)
Hue        < 0.785  to the left,  improve=20.11866, (0 missing)
Surrogate splits:
  Flavanoids      < 1.48   to the left,  agree=0.899, adj=0.758, (0 split)
Colour          < 4.79   to the right, agree=0.886, adj=0.727, (0 split)
Hue             < 0.765  to the left,  agree=0.835, adj=0.606, (0 split)
Alcohol         < 12.745 to the right, agree=0.797, adj=0.515, (0 split)
Proanthocyanins < 1.285  to the left,  agree=0.772, adj=0.455, (0 split)

Node number 4: 40 observations
predicted class=one    expected loss=0.025  P(node) =0.3174603
class counts:    39     0     1
probabilities: 0.975 0.000 0.025

Node number 5: 7 observations
predicted class=two    expected loss=0.5714286  P(node) =0.05555556
class counts:     2     2     3
probabilities: 0.286 0.286 0.429

Node number 6: 33 observations
predicted class=three  expected loss=0.09090909  P(node) =0.2619048
class counts:     0    30     3
probabilities: 0.000 0.909 0.091

Node number 7: 46 observations
predicted class=two    expected loss=0.06521739  P(node) =0.3650794
class counts:     1     2    43
probabilities: 0.022 0.043 0.935

###################################
# Confusion Matrix and Statistics
```

```
##############################

# train data

Reference
Prediction one three two
one     39     0   1
three    0    30   3
two      3     4  46

Overall Statistics

Accuracy : 0.9127
95% CI : (0.8492, 0.9556)
No Information Rate : 0.3968
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.867

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: one Class: three Class: two
Sensitivity              0.9286       0.8824     0.9200
Specificity              0.9881       0.9674     0.9079
Pos Pred Value           0.9750       0.9091     0.8679
Neg Pred Value           0.9651       0.9570     0.9452
Prevalence               0.3333       0.2698     0.3968
Detection Rate           0.3095       0.2381     0.3651
Detection Prevalence     0.3175       0.2619     0.4206
Balanced Accuracy        0.9583       0.9249     0.9139

# test data

          Reference
Prediction one three two
one     17     0   0
three    0     7   2
two      0     7  19

Overall Statistics

Accuracy : 0.8269
95% CI : (0.6967, 0.9177)
No Information Rate : 0.4038
P-Value [Acc > NIR] : 4.728e-10
```

```
Kappa : 0.7315

Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: one Class: three Class: two
Sensitivity              1.0000       0.5000     0.9048
Specificity              1.0000       0.9474     0.7742
Pos Pred Value           1.0000       0.7778     0.7308
Neg Pred Value           1.0000       0.8372     0.9231
Prevalence               0.3269       0.2692     0.4038
Detection Rate           0.3269       0.1346     0.3654
Detection Prevalence     0.3269       0.1731     0.5000
Balanced Accuracy        1.0000       0.7237     0.8395

###################################
# PCA: CART
###################################

Call:
  rpart(formula = Type ~ ., data = trainPCA1)
n= 126

CP nsplit  rel error    xerror       xstd
1 0.5000000      0 1.00000000 1.0000000 0.07225916
2 0.4473684      1 0.50000000 0.5526316 0.06962503
3 0.0100000      2 0.05263158 0.1184211 0.03803779

Variable importance
PC2 PC1 PC7 PC6 PC4 PC3 PC5
39  29  10   7   7   4   4

Node number 1: 126 observations,    complexity param=0.5
predicted class=two    expected loss=0.6031746  P(node) =1
class counts:    42    34    50
probabilities: 0.333 0.270 0.397
left son=2 (78 obs) right son=3 (48 obs)
Primary splits:
  PC2 < -0.6379261 to the right, improve=39.512970, (0 missing)
PC1 < 1.797617   to the right, improve=30.065260, (0 missing)
PC7 < 1.035708   to the left,  improve= 4.679042, (0 missing)
PC6 < 0.1657563  to the right, improve= 3.948413, (0 missing)
PC3 < -1.474407  to the right, improve= 3.531136, (0 missing)
Surrogate splits:
  PC3 < -1.474407  to the right, agree=0.675, adj=0.146, (0 split)
PC7 < 1.035708   to the left,  agree=0.667, adj=0.125, (0 split)
PC4 < -1.453536  to the right, agree=0.643, adj=0.063, (0 split)
```

```
PC1 < -1.057591  to the left,  agree=0.635, adj=0.042, (0 split)
PC6 < -0.8596221 to the right, agree=0.635, adj=0.042, (0 split)

Node number 2: 78 observations,    complexity param=0.4473684
predicted class=one    expected loss=0.474359  P(node) =0.6190476
class counts:    41    34     3
probabilities: 0.526 0.436 0.038
left son=4 (44 obs) right son=5 (34 obs)
Primary splits:
  PC1 < 0.7938279  to the left,  improve=35.921910, (0 missing)
PC2 < 1.894848   to the left,  improve= 5.552999, (0 missing)
PC5 < 1.027212   to the left,  improve= 5.328289, (0 missing)
PC6 < -0.8160478 to the right, improve= 4.952434, (0 missing)
PC7 < -0.899874  to the right, improve= 3.850985, (0 missing)
Surrogate splits:
  PC2 < 1.894848   to the left,  agree=0.692, adj=0.294, (0 split)
PC7 < -0.687789  to the right, agree=0.667, adj=0.235, (0 split)
PC6 < -0.8160478 to the right, agree=0.654, adj=0.206, (0 split)
PC4 < -0.5639463 to the right, agree=0.641, adj=0.176, (0 split)
PC5 < 1.027212   to the left,  agree=0.628, adj=0.147, (0 split)

Node number 3: 48 observations
predicted class=two    expected loss=0.02083333  P(node) =0.3809524
class counts:     1     0    47
probabilities: 0.021 0.000 0.979

Node number 4: 44 observations
predicted class=one    expected loss=0.06818182  P(node) =0.3492063
class counts:    41     0     3
probabilities: 0.932 0.000 0.068

Node number 5: 34 observations
predicted class=three  expected loss=0  P(node) =0.2698413
class counts:     0    34     0
probabilities: 0.000 1.000 0.000


###############################
# Confusion Matrix and Statistics
###############################

# train data

Reference
Prediction one three two
one     41     0   3
three    0    34   0
two      1     0  47
```

```
Overall Statistics

Accuracy : 0.9683
95% CI : (0.9207, 0.9913)
No Information Rate : 0.3968
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9519

Mcnemar's Test P-Value : NA

Statistics by Class:

Class: one Class: three Class: two
Sensitivity                 0.9762       1.0000     0.9400
Specificity                 0.9643       1.0000     0.9868
Pos Pred Value              0.9318       1.0000     0.9792
Neg Pred Value              0.9878       1.0000     0.9615
Prevalence                  0.3333       0.2698     0.3968
Detection Rate              0.3254       0.2698     0.3730
Detection Prevalence        0.3492       0.2698     0.3810
Balanced Accuracy           0.9702       1.0000     0.9634

# test data

Reference
Prediction one three two
one     17     0   2
three    0    14   2
two      0     0  17

Overall Statistics

Accuracy : 0.9231
95% CI : (0.8146, 0.9786)
No Information Rate : 0.4038
P-Value [Acc > NIR] : 4.536e-15

Kappa : 0.8844

Mcnemar's Test P-Value : NA

Statistics by Class:

  Class: one Class: three Class: two
Sensitivity                 1.0000       1.0000     0.8095
Specificity                 0.9429       0.9474     1.0000
```

```
Pos Pred Value              0.8947       0.8750      1.0000
Neg Pred Value              1.0000       1.0000      0.8857
Prevalence                  0.3269       0.2692      0.4038
Detection Rate              0.3269       0.2692      0.3269
Detection Prevalence        0.3654       0.3077      0.3269
Balanced Accuracy           0.9714       0.9737      0.9048

################################
# LDA: CART
################################

Call:
  rpart(formula = Type ~ ., data = trainLDA)
n= 126

CP nsplit rel error    xerror       xstd
1 0.5526316      0 1.0000000 1.0000000 0.07225916
2 0.4473684      1 0.4473684 0.4473684 0.06555933
3 0.0100000      2 0.0000000 0.0000000 0.00000000

Variable importance
LD1 LD2
50  50

Node number 1: 126 observations,    complexity param=0.5526316
predicted class=two    expected loss=0.6031746  P(node) =1
class counts:    42    34    50
probabilities: 0.333 0.270 0.397
left son=2 (76 obs) right son=3 (50 obs)
Primary splits:
  LD2 < -14.14982 to the left,  improve=45.40518, (0 missing)
LD1 < -7.107402 to the right, improve=37.33195, (0 missing)
Surrogate splits:
  LD1 < -11.3388  to the left,  agree=0.698, adj=0.24, (0 split)

Node number 2: 76 observations,    complexity param=0.4473684
predicted class=one    expected loss=0.4473684  P(node) =0.6031746
class counts:    42    34     0
probabilities: 0.553 0.447 0.000
left son=4 (42 obs) right son=5 (34 obs)
Primary splits:
  LD1 < -8.722805 to the left,  improve=37.5789500, (0 missing)
LD2 < -17.57283 to the right, improve= 0.5642415, (0 missing)
Surrogate splits:
  LD2 < -17.66692 to the right, agree=0.579, adj=0.059, (0 split)

Node number 3: 50 observations
predicted class=two    expected loss=0  P(node) =0.3968254
```

```
class counts:     0     0    50
probabilities: 0.000 0.000 1.000

Node number 4: 42 observations
predicted class=one    expected loss=0  P(node) =0.3333333
class counts:    42     0     0
probabilities: 1.000 0.000 0.000

Node number 5: 34 observations
predicted class=three  expected loss=0  P(node) =0.2698413
class counts:     0    34     0
probabilities: 0.000 1.000 0.000

##############################
# Confusion Matrix and Statistics
##############################

# train data

Reference
Prediction one three two
one     42     0   0
three    0    34   0
two      0     0  50

Overall Statistics

Accuracy : 1
95% CI : (0.9711, 1)
No Information Rate : 0.3968
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 1

Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: one Class: three Class: two
Sensitivity              1.0000       1.0000     1.0000
Specificity              1.0000       1.0000     1.0000
Pos Pred Value           1.0000       1.0000     1.0000
Neg Pred Value           1.0000       1.0000     1.0000
Prevalence               0.3333       0.2698     0.3968
Detection Rate           0.3333       0.2698     0.3968
Detection Prevalence     0.3333       0.2698     0.3968
Balanced Accuracy        1.0000       1.0000     1.0000
```

```
# test data

          Reference
Prediction one three two
one     17     0   1
three    0    14   0
two      0     0  20

Overall Statistics

Accuracy : 0.9808
95% CI : (0.8974, 0.9995)
No Information Rate : 0.4038
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9708

Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: one Class: three Class: two
Sensitivity              1.0000       1.0000     0.9524
Specificity              0.9714       1.0000     1.0000
Pos Pred Value           0.9444       1.0000     1.0000
Neg Pred Value           1.0000       1.0000     0.9687
Prevalence               0.3269       0.2692     0.4038
Detection Rate           0.3269       0.2692     0.3846
Detection Prevalence     0.3462       0.2692     0.3846
Balanced Accuracy        0.9857       1.0000     0.9762

################################
# Original data with repeated CV: CART
################################

CART

178 samples
13 predictor
3 classes: 'one', 'three', 'two'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (7 fold, repeated 10 times)
Summary of sample sizes: 152, 152, 153, 153, 153, 152, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
0.05607477  0.8599601  0.7874144
```

```
0.31775701   0.7904699   0.6769536
0.49532710   0.4844801   0.1537125

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.05607477.

##############################
# PCA with repeated CV: CART
##############################

CART

178 samples
13 predictor
3 classes: 'one', 'three', 'two'

No pre-processing
Resampling: Cross-Validated (7 fold, repeated 10 times)
Summary of sample sizes: 153, 153, 153, 153, 152, 152, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
0.0000000   0.9508602   0.9256554
0.4485981   0.8157393   0.7166421
0.4859813   0.5306054   0.2410515

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.

##############################
# LDA with repeated CV: CART
##############################

CART

178 samples
2 predictor
3 classes: 'one', 'three', 'two'

No pre-processing
Resampling: Cross-Validated (7 fold, repeated 10 times)
Summary of sample sizes: 153, 152, 152, 153, 152, 153, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
0.0000000   0.9943718   0.9914876
0.4485981   0.8554487   0.7768710
0.5420561   0.5816127   0.3267775
```

```
Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.

###################################
###################################
```

Listing A.2: R Output: Wine Data

A.7 R Code for the Number of PC Investigation

The R script for the investigation conducted to find out the best number of principal components to use to build the optimal model for the wine data set example in chapter 2 is as follows

```
#get principal components
mydataPCA <- data.frame(Type = mydata$Type, prin_comp$x)

trainPCA <- mydataPCA[indexes,]
testPCA <- mydataPCA[-indexes,]

#create data frame for results
NumberofPC <- 2:13
accuracyRate <- 2:13
WineDataPC <- data.frame(NumberofPC, accuracyRate)

ctrl.1 <- trainControl(method = 'repeatedcv', number = 7, repeats = 10)

#####################################
#####################################

#we are interested in first 13 PCs
trainPCA1 <- trainPCA[,1:14]
testPCA1 <- testPCA[,1:14]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[12] <-  treeModelexp$results[1,2]

#####################################
#####################################

#we are interested in first 12 PCs
trainPCA1 <- trainPCA1[,1:13]
testPCA1 <- testPCA1[,1:13]
```

```
treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[11] <-  treeModelexp$results[1,2]

#####################################
#####################################

#we are interested in first 11 PCs
trainPCA1 <- trainPCA1[,1:12]
testPCA1 <- testPCA1[,1:12]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[10] <-  treeModelexp$results[1,2]

#####################################
#####################################

#we are interested in first 10 PCs
trainPCA1 <- trainPCA1[,1:11]
testPCA1 <- testPCA1[,1:11]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[9] <-  treeModelexp$results[1,2]

#####################################
#####################################

#we are interested in first 9 PCs
trainPCA1 <- trainPCA1[,1:10]
testPCA1 <- testPCA1[,1:10]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[8] <-  treeModelexp$results[1,2]

#####################################
#####################################
```

```
#we are interested in first 8 PCs
trainPCA1 <- trainPCA1[,1:9]
testPCA1 <- testPCA1[,1:9]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[7] <-  treeModelexp$results[1,2]

#####################################
#####################################

#we are interested in first 7 PCs
trainPCA1 <- trainPCA1[,1:8]
testPCA1 <- testPCA1[,1:8]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[6] <-  treeModelexp$results[1,2]

#####################################
#####################################

#we are interested in first 6 PCs
trainPCA1 <- trainPCA1[,1:7]
testPCA1 <- testPCA1[,1:7]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[5] <-  treeModelexp$results[1,2]

#####################################
#####################################

#we are interested in first 5 PCs
trainPCA1 <- trainPCA1[,1:6]
testPCA1 <- testPCA1[,1:6]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
```

```
WineDataPC$accuracyRate[4] <-  treeModelexp$results[1,2]

###################################
###################################

#we are interested in first 4 PCs
trainPCA1 <- trainPCA1[,1:5]
testPCA1 <- testPCA1[,1:5]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[3] <-  treeModelexp$results[1,2]

###################################
###################################

#we are interested in first 3 PCs
trainPCA1 <- trainPCA1[,1:4]
testPCA1 <- testPCA1[,1:4]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[2] <-  treeModelexp$results[1,2]

###################################
###################################

#we are interested in first 2 PCs
trainPCA1 <- trainPCA1[,1:3]
testPCA1 <- testPCA1[,1:3]

treeModelexp = train(Type ~ ., data=trainPCA1, method="rpart", preProcess = c(
    "center", "scale"), trControl = ctrl.1)

#show results
WineDataPC$accuracyRate[1] <-  treeModelexp$results[1,2]

###################################
###################################

plot(WineDataPC$NumberofPC, WineDataPC$accuracyRate, xlab = 'Number of
    Principal Components', ylab = 'Accuracy rate')
lines(WineDataPC$NumberofPC, WineDataPC$accuracyRate)
```

```
####################################
####################################
```

Listing A.3: Wine Data: PCA Investigation

Appendix B

R Code and Output: Chapter 3

B.1 R Code for the Wine Data Neural Network

The R script for the neural network model built for wine data set example in chapter 3 is as follows

```
rm(list=ls())
library(caret)

# read in data
mydata <- read.csv("wine.csv", sep = ",", header = FALSE)
str(mydata)

# is there any missing data?
apply(mydata,2,function(x) sum(is.na(x)))

# changing variable names
colnames(mydata) <- c('Type', 'Alcohol', 'Malic', 'Ash', 'Alcalinity', '
    Magnesium', 'Phenols', 'Flavanoids', 'Nonflavanoids', 'Proanthocyanins', '
    Colour', 'Hue', 'Dilution', 'Proline')

# changing class names
indexes <- which(mydata$Type == 1)
mydata$Type[indexes] <- 'one'
indexes <- which(mydata$Type == 2)
mydata$Type[indexes] <- 'two'
indexes <- which(mydata$Type == 3)
mydata$Type[indexes] <- 'three'

# changing response variable to factor
mydata$Type <- as.factor(mydata$Type)

#####################################
# Use a neural network to classify the wine type
```

```
# Divide data set into train and test set: 70/30
# Fit to the original data
#####################################

load("indexesWine.Rdata")

train <- mydata[indexes,]
test <- mydata[-indexes,]

# train the model

ctrl.1 <- trainControl(method = "none")

nnEX <- train(Type ~ ., data=train, method = 'nnet', metric = 'Accuracy',
    tuneGrid = expand.grid(size = 3, decay = 0.5), trace = FALSE, preProcess =
    c("center", "scale"), trControl=ctrl.1, maxit = 100)

confusionMatrix(predict(nnEX, newdata=train[,-1]), train[,1])
confusionMatrix(predict(nnEX, newdata=test[,-1]), test[,1])

# load package to plot model
library(devtools)
source_url('https://gist.githubusercontent.com/Peque/41a9e20d6687f2f3108d/raw/
    85e14f3a292e126f1454864427e3a189c2fe33f3/nnet_plot_update.r')

#plot model
plot.nnet(nnEX)

confusionMatrix(predict(nnEX, newdata=train[,-1]), train[,1])
confusionMatrix(predict(nnEX, newdata=test[,-1]), test[,1])

#####################################
# 7-fold cross validation, repeated 10 times on the original data
# loading model package
#####################################

ctrl.1 <- trainControl(method = 'repeatedcv', number = 7, repeats = 10)

nnEXcv <- train(Type ~ ., data=mydata, method = 'nnet', metric = 'Accuracy',
    tuneGrid = expand.grid(size = 3, decay = 0.5), trace = FALSE, preProcess =
    c("center", "scale"), trControl=ctrl.1, maxit = 100)

#####################################
#####################################
```

Listing B.1: Wine Data NN R Script

B.2 R Output for the Wine Data Neural Network

The R script for the output of the neural network model built for wine data set example in chapter 3 is as follows

```
###############################################
# Original data: CART (NEURAL NETWORK)
###############################################

Neural Network

126 samples
13 predictor
3 classes: 'one', 'three', 'two'

Pre-processing: centered (13), scaled (13)
Resampling: None

# summary of model

a 13-3-3 network with 54 weights
options were - softmax modelling  decay=0.5
b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1  i9->h1
-0.84   0.36   0.21   0.21   0.26   0.14  -0.10  -0.75   0.14  -0.27
i10->h1 i11->h1 i12->h1 i13->h1
0.87  -0.61  -0.69  -0.09
b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2
-0.52  -1.09  -0.30  -0.77   0.48  -0.20  -0.20   0.17  -0.04   0.34
i10->h2 i11->h2 i12->h2 i13->h2
-0.92   0.74   0.12  -1.04
b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3  i9->h3
-0.53   0.74   0.21   0.60  -0.87   0.09   0.25   0.62  -0.08  -0.11
i10->h3 i11->h3 i12->h3 i13->h3
0.01  -0.05   0.68   1.06
b->o1 h1->o1 h2->o1 h3->o1
0.02  -1.22  -1.37   2.58
b->o2 h1->o2 h2->o2 h3->o2
-0.06   2.43  -1.37  -1.26
b->o3 h1->o3 h2->o3 h3->o3
0.05  -1.21   2.74  -1.31

###############################################
# Confusion Matrix and Statistics
###############################################

# train data

Confusion Matrix and Statistics
```

```
Reference
Prediction one three two
one     42     0   0
three    0    34   0
two      0     0  50

Overall Statistics

Accuracy : 1
95% CI : (0.9711, 1)
No Information Rate : 0.3968
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 1

Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: one Class: three Class: two
Sensitivity              1.0000       1.0000     1.0000
Specificity              1.0000       1.0000     1.0000
Pos Pred Value           1.0000       1.0000     1.0000
Neg Pred Value           1.0000       1.0000     1.0000
Prevalence               0.3333       0.2698     0.3968
Detection Rate           0.3333       0.2698     0.3968
Detection Prevalence     0.3333       0.2698     0.3968
Balanced Accuracy        1.0000       1.0000     1.0000

# test data

Reference
Prediction one three two
one     17     0   0
three    0    14   2
two      0     0  19

Overall Statistics

Accuracy : 0.9615
95% CI : (0.8679, 0.9953)
No Information Rate : 0.4038
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.942

Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

Class: one Class: three Class: two
Sensitivity                 1.0000       1.0000     0.9048
Specificity                 1.0000       0.9474     1.0000
Pos Pred Value              1.0000       0.8750     1.0000
Neg Pred Value              1.0000       1.0000     0.9394
Prevalence                  0.3269       0.2692     0.4038
Detection Rate              0.3269       0.2692     0.3654
Detection Prevalence        0.3269       0.3077     0.3654
Balanced Accuracy           1.0000       0.9737     0.9524

##############################################
# Original data with repeated CV: CART (NEURAL NETWORK)
##############################################

Neural Network

178 samples
13 predictor
3 classes: 'one', 'three', 'two'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (7 fold, repeated 10 times)
Summary of sample sizes: 153, 152, 153, 152, 152, 153, ...
Resampling results:

Accuracy   Kappa
0.9813791  0.9718137

Tuning parameter 'size' was held constant at a value of 3
Tuning parameter
'decay' was held constant at a value of 0.5

##############################################
##############################################
```

Listing B.2: Wine Data: R Output

Appendix C

R Code and Output: Chapter 4

C.1 First Ten Observations: The Iris Data

The output of the first ten observation of the subsetted Iris data set in chapter 4 is as follows

Sepal Length	Sepal Width	Species
5.1	3.5	*setosa*
4.9	3.0	*setosa*
4.7	3.2	*setosa*
4.6	3.1	*setosa*
5.0	3.6	*setosa*
5.4	3.9	*setosa*
4.6	3.4	*setosa*
5.0	3.4	*setosa*
4.4	2.9	*setosa*
4.9	3.1	*setosa*

Table C.1: First ten observations of the subsetted Iris data set.

C.2 R Code for the Iris Data Support Vector Machine

The R script for the support vector machine model built for the subset of the Iris data set example in chapter 4 is as follows

```
rm(list=ls())
library(caret)
library(datasets)
library(kernlab)

#####################################
# read in data
#####################################
```

```
data(iris)

#subset the data
mydata <- iris[1:100,c(1,2,5)]
str(mydata)

# Is there any missing data?
apply(mydata,2,function(x) sum(is.na(x)))

#have a look at the first ten observation
head(mydata, n = 10)

# simple summary statistics
summary(mydata)

#removing unused class in response variable
mydata$Species <- factor(mydata$Species, levels = c('setosa', 'versicolor'))

#distribution of classes
cbind(freq = table(mydata$Species), percentage = prop.table(table(mydata$
    Species))*100)

#box plot of feature variables
boxplot(mydata[,1:2])

####################################
# Use a support vector machine to classify the species type
# Divide data set into train and test set: 70/30
# Fit to the original data
####################################

# indexes <- createDataPartition(mydata$Species, times = 1, p = 0.7, list =
    FALSE)
# save(indexes, file="E://Project//Rscript//indexesIris.Rdata")

load("indexesIris.Rdata")

train <- mydata[indexes,]
test <- mydata[-indexes,]

#build model using kernlab function
svmLinEX <- ksvm(Species ~ ., data = train, type="C-svc", scale = TRUE, kernel
    ="vanilladot")

summary(svmLinEX)

#use model to predict unseen observations
```

```
preds <- predict(svmLinEX, test[,-3])
confusionMatrix(preds, test$Species)

#plot svm using train data set
kernlab::plot(svmLinEX, data = train)

#### plot decision boundary on the data set. Lines not showing####
#plot(mydata[,1:2], col=mydata[,3], pch=1, xlab="", ylab="")
#w <- colSums(coef(svmLinEX)[[1]] * mydata[,1:2][unlist(alphaindex(svmLinEX))
    ,])
#b <- b(svmLinEX)
#abline(a = b/w[1], b = -w[2]/w[1], col = 'black')
#abline(a = (b+1)/w[1], b = -w[2]/w[1],lty=2, col = 'black')
#abline(a = (b-1)/w[1], b = -w[2]/w[1],lty=2, col = 'black')

#####################################
# 7-fold cross validation, repeated 10 times on the original data
# loading model package
#####################################

ctrl.1 <- trainControl(method = 'repeatedcv', number = 7, repeats = 10)

svmLinEXcv <- train(Species ~ ., data = mydata, method = 'svmLinear', metric =
     'Accuracy', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

#####################################
#####################################
```

Listing C.1: SVM: Iris Data

C.3 R Output for the Iris Data Support Vector Machine

The R script for the output of the support vector machine model built for the subset of the Iris data set example in chapter 4 is as follows

```
#############################################
#CART (SUPPORT VECTOR MACHINE) original data set
#############################################

Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
parameter : cost C = 1
```

```
Linear (vanilla) kernel function.

Number of Support Vectors : 9

Objective Function Value : -5.461
Training error : 0

# summary

Length  Class   Mode
1   ksvm    S4

##########################################
# Confusion Matrix and Statistics
##########################################

Confusion Matrix and Statistics

Reference
Prediction    setosa versicolor
setosa          15          0
versicolor       0         15

Accuracy : 1
95% CI : (0.8843, 1)
No Information Rate : 0.5
P-Value [Acc > NIR] : 9.313e-10

Kappa : 1

Mcnemar's Test P-Value : NA

Sensitivity : 1.0
Specificity : 1.0
Pos Pred Value : 1.0
Neg Pred Value : 1.0
Prevalence : 0.5
Detection Rate : 0.5
Detection Prevalence : 0.5
Balanced Accuracy : 1.0

'Positive' Class : setosa

#####################################
#CART (SUPPORT VECTOR MACHINE) original data set with repeated cross
    validation
#####################################
```

```
Support Vector Machines with Linear Kernel

100 samples
2 predictor
2 classes: 'setosa', 'versicolor'

Pre-processing: centered (2), scaled (2)
Resampling: Cross-Validated (7 fold, repeated 10 times)
Summary of sample sizes: 85, 86, 86, 86, 86, 86, ...
Resampling results across tuning parameters:

C     Accuracy   Kappa
0.10  0.9897959  0.9795918
0.25  0.9908163  0.9816327
0.50  0.9897959  0.9795918
0.75  0.9897959  0.9795918
1.00  0.9908163  0.9816327

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 0.25.

########################################
########################################
```

Listing C.2: Iris Data: SVM Output

Appendix D

R Code and Output: Default in Payment Data Set

D.1 First Ten Observations: The Credit Data

The first ten observation of the credit data set as discussed in chapter 5 are shown on the following page.

Obs	LIMITBAL	SEX	EDUCATION	MARRIAGE	AGE	PAY0	PAY2	PAY3	PAY4	PAY5	PAY6
1	20000	2	2	1	24	2	2	−1	−1	−2	−2
2	120000	2	2	2	26	−1	2	0	0	0	2
3	90000	2	2	2	34	0	0	0	0	0	0
4	50000	2	2	1	37	0	0	0	0	0	0
5	50000	1	2	1	57	−1	0	−1	0	0	0
6	50000	1	1	2	37	0	0	0	0	0	0
7	500000	1	1	2	29	0	0	0	0	0	0
8	100000	2	2	2	23	0	−1	−1	0	0	−1
9	140000	2	3	1	28	0	0	2	0	0	0
10	20000	1	3	2	35	−2	−2	−2	−2	−1	−1

Obs	BILLAMT1	BILLAMT2	BILLAMT3	BILLAMT4	BILLAMT5	BILLAMT6
1	3913	3102	689	0	0	0
2	2682	1725	2682	3272	3455	3261
3	29239	14027	13559	14331	14948	15549
4	46990	48233	49291	28314	28959	29547
5	8617	5670	35835	20940	19146	19131
6	64400	57069	57608	19394	19619	20024
7	367965	412023	445007	542653	483003	473944
8	11876	380	601	221	−159	567
9	11285	14096	12108	12211	11793	3719
10	0	0	0	0	13007	13912

Obs	PAYAMT1	PAYAMT2	PAYAMT3	PAYAMT4	PAYAMT5	PAYAMT6	defaultPayment
1	0	689	0	0	0	0	yes
2	0	1000	1000	1000	0	2000	yes
3	1518	1500	1000	1000	1000	5000	no
4	2000	2019	1200	1100	1069	1000	no
5	2000	36681	10000	9000	689	679	no
6	2500	1815	657	1000	1000	800	no
7	55000	40000	38000	20239	13750	13770	no
8	380	601	0	581	1687	1542	no
9	3329	0	432	1000	1000	1000	no
10	0	0	0	13007	1122	0	no

Table D.1: First ten observations of the credit data set.

D.2 Summary of these Data

Summary statistics for the credit data discussed in chapter 5 are as follows:

	LIMITBAL	SEX	EDUCATION	MARRIAGE	AGE	PAY0	PAY2	PAY3	PAY4	PAY5	PAY6
Min	10000	1.000	0.000	0.000	21.00	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000
Q_1	50000	1.000	1.000	1.000	28.00	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000	-1.0000
$\tilde{x}$	140000	2.000	2.000	2.000	34.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
$\overline{x}$	167484	1.604	1.853	1.552	35.49	-0.0167	-0.1338	-0.1662	-0.2207	-0.2662	-0.2911
Q_3	240000	2.000	2.000	2.000	41.00	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Max	1000000	2.000	6.000	3.000	79.00	8.0000	8.0000	8.0000	8.0000	8.0000	8.0000

	BILLAMT1	BILLAMT2	BILLAMT3	BILLAMT4	BILLAMT5	BILLAMT6
Min	-165580	-69777	-157264	-170000	-81334	-339603
Q_1	3559	2985	2666	2327	1763	1256
$\tilde{x}$	22382	21200	20089	19052	18105	17071
$\overline{x}$	51223	49179	47013	43263	40311	38872
Q_3	67091	64006	60165	54506	50191	49198
Max	964511	983931	1664089	891586	927171	961664

	PAYAMT1	PAYAMT2	PAYAMT3	PAYAMT4	PAYAMT5	PAYAMT6
Min	0	0	0	0	0.0	0.0
Q_1	1000	833	390	296	252.5	117.8
$\tilde{x}$	2100	2009	1800	1500	1500.0	1500.0
$\overline{x}$	5664	5921	5226	4826	4799.4	5215.5
Q_3	5006	5000	4505	4013	4031.5	4000.0
Max	873552	1684259	896040	621000	426529.0	528666.0

Table D.2: Summary statistics of the variables in the credit data set.

D.3 Variance-Covariance Matrix

The variance-covariance matrix of the credit data discussed in chapter 5 is as follows:

	LIMITBAL	SEX	EDUCATION	MARRIAGE	AGE	PAY0	PAY2	PAY3	PAY4	PAY5	PAY6
LIMITBAL	1.0000	0.0248	−0.2192	−0.1081	0.1447	−0.2712	−0.2964	−0.2861	−0.2675	−0.2494	−0.2352
SEX	0.0248	1.0000	0.0142	−0.0314	−0.0909	−0.0576	−0.0708	−0.0661	−0.0602	−0.0551	−0.0440
EDUCATION	−0.2192	0.0142	1.0000	−0.1435	0.1751	0.1054	0.1216	0.1140	0.1088	0.0975	0.0823
MARRIAGE	−0.1081	−0.0314	−0.1435	1.0000	−0.4142	0.0199	0.0242	0.0327	0.0331	0.0356	0.0343
AGE	0.1447	−0.0909	0.1751	−0.4142	1.0000	−0.0394	−0.0501	−0.0530	−0.0497	−0.0538	−0.0488
PAY0	−0.2712	−0.0576	0.1054	0.0199	−0.0394	1.0000	0.6722	0.5742	0.5388	0.5094	0.4746
PAY2	−0.2964	−0.0708	0.1216	0.0242	−0.0501	0.6722	1.0000	0.7666	0.6621	0.6228	0.5755
PAY3	−0.2861	−0.0661	0.1140	0.0327	−0.0530	0.5742	0.7666	1.0000	0.7774	0.6868	0.6327
PAY4	−0.2675	−0.0602	0.1088	0.0331	−0.0497	0.5388	0.6621	0.7774	1.0000	0.8198	0.7164
PAY5	−0.2494	−0.0551	0.0975	0.0356	−0.0538	0.5094	0.6228	0.6868	0.8198	1.0000	0.8169
PAY6	−0.2352	−0.0440	0.0823	0.0343	−0.0488	0.4746	0.5755	0.6327	0.7164	0.8169	1.0000
BILLAMT1	0.2854	−0.0336	0.0236	−0.0235	0.0562	0.1871	0.2349	0.2085	0.2028	0.2067	0.2074
BILLAMT2	0.2783	−0.0312	0.0187	−0.0216	0.0543	0.1899	0.2353	0.2373	0.2258	0.2269	0.2269
BILLAMT3	0.2832	−0.0246	0.0130	−0.0249	0.0537	0.1798	0.2241	0.2275	0.2450	0.2433	0.2412
BILLAMT4	0.2940	−0.0219	−0.0005	−0.0233	0.0514	0.1791	0.2222	0.2272	0.2459	0.2719	0.2664
BILLAMT5	0.2956	−0.0170	−0.0076	−0.0254	0.0493	0.1806	0.2213	0.2251	0.2429	0.2698	0.2909
BILLAMT6	0.2904	−0.0167	−0.0091	−0.0212	0.0476	0.1770	0.2194	0.2223	0.2392	0.2625	0.2851
PAYAMT1	0.1952	−0.0002	−0.0375	−0.0060	0.0261	−0.0793	−0.0807	0.0013	−0.0094	−0.0061	−0.0015
PAYAMT2	0.1784	−0.0014	−0.0300	−0.0081	0.0218	−0.0701	−0.0590	−0.0668	−0.0019	−0.0032	−0.0052
PAYAMT3	0.2102	−0.0086	−0.0399	−0.0035	0.0292	−0.0706	−0.0559	−0.0533	−0.0692	0.0091	0.0058
PAYAMT4	0.2032	−0.0022	−0.0382	−0.0127	0.0214	−0.0640	−0.0469	−0.0461	−0.0435	−0.0583	0.0190
PAYAMT5	0.2172	−0.0017	−0.0404	−0.0012	0.0228	−0.0582	−0.0371	−0.0359	−0.0336	−0.0333	−0.0464
PAYAMT6	0.2196	−0.0028	−0.0372	−0.0066	0.0195	−0.0587	−0.0365	−0.0359	−0.0266	−0.0230	−0.0253

Table D.3: Variance-covariance matrix of the credit data set: The first eleven variables.

	BILLAMT1	BILLAMT2	BILLAMT3	BILLAMT4	BILLAMT5	BILLAMT6
LIMITBAL	0.2854	0.2783	0.2832	0.2940	0.2956	0.2904
SEX	−0.0336	−0.0312	−0.0246	−0.0219	−0.0170	−0.0167
EDUCATION	0.0236	0.0187	0.0130	−0.0005	−0.0076	−0.0091
MARRIAGE	−0.0235	−0.0216	−0.0249	−0.0233	−0.0254	−0.0212
AGE	0.0562	0.0543	0.0537	0.0514	0.0493	0.0476
PAY0	0.1871	0.1899	0.1798	0.1791	0.1806	0.1770
PAY2	0.2349	0.2353	0.2241	0.2222	0.2213	0.2194
PAY3	0.2085	0.2373	0.2275	0.2272	0.2251	0.2223
PAY4	0.2028	0.2258	0.2450	0.2459	0.2429	0.2392
PAY5	0.2067	0.2269	0.2433	0.2719	0.2698	0.2625
PAY6	0.2074	0.2269	0.2412	0.2664	0.2909	0.2851
BILLAMT1	1.0000	0.9515	0.8923	0.8603	0.8298	0.8027
BILLAMT2	0.9515	1.0000	0.9283	0.8925	0.8598	0.8316
BILLAMT3	0.8923	0.9283	1.0000	0.9240	0.8839	0.8533
BILLAMT4	0.8603	0.8925	0.9240	1.0000	0.9401	0.9009
BILLAMT5	0.8298	0.8598	0.8839	0.9401	1.0000	0.9462
BILLAMT6	0.8027	0.8316	0.8533	0.9009	0.9462	1.0000
PAYAMT1	0.1403	0.2804	0.2443	0.2330	0.2170	0.2000
PAYAMT2	0.0994	0.1009	0.3169	0.2076	0.1812	0.1727
PAYAMT3	0.1569	0.1507	0.1300	0.3000	0.2523	0.2338
PAYAMT4	0.1583	0.1474	0.1434	0.1302	0.2931	0.2502
PAYAMT5	0.1670	0.1580	0.1797	0.1604	0.1416	0.3077
PAYAMT6	0.1793	0.1743	0.1823	0.1776	0.1642	0.1155

Table D.4: Variance-covariance matrix of the credit data set: Variables twelve to seventeen.

	PAYAMT1	PAYAMT2	PAYAMT3	PAYAMT4	PAYAMT5	PAYAMT6
LIMITBAL	0.1952	0.1784	0.2102	0.2032	0.2172	0.2196
SEX	−0.0002	−0.0014	−0.0086	−0.0022	−0.0017	−0.0028
EDUCATION	−0.0375	−0.0300	−0.0399	−0.0382	−0.0404	−0.0372
MARRIAGE	−0.0060	−0.0081	−0.0035	−0.0127	−0.0012	−0.0066
AGE	0.0261	0.0218	0.0292	0.0214	0.0228	0.0195
PAY0	−0.0793	−0.0701	−0.0706	−0.0640	−0.0582	−0.0587
PAY2	−0.0807	−0.0590	−0.0559	−0.0469	−0.0371	−0.0365
PAY3	0.0013	−0.0668	−0.0533	−0.0461	−0.0359	−0.0359
PAY4	−0.0094	−0.0019	−0.0692	−0.0435	−0.0336	−0.0266
PAY5	−0.0061	−0.0032	0.0091	−0.0583	−0.0333	−0.0230
PAY6	−0.0015	−0.0052	0.0058	0.0190	−0.0464	−0.0253
BILLAMT1	0.1403	0.0994	0.1569	0.1583	0.1670	0.1793
BILLAMT2	0.2804	0.1009	0.1507	0.1474	0.1580	0.1743
BILLAMT3	0.2443	0.3169	0.1300	0.1434	0.1797	0.1823
BILLAMT4	0.2330	0.2076	0.3000	0.1302	0.1604	0.1776
BILLAMT5	0.2170	0.1812	0.2523	0.2931	0.1416	0.1642
BILLAMT6	0.2000	0.1727	0.2338	0.2502	0.3077	0.1155
PAYAMT1	1.0000	0.2856	0.2522	0.1996	0.1485	0.1857
PAYAMT2	0.2856	1.0000	0.2448	0.1801	0.1809	0.1576
PAYAMT3	0.2522	0.2448	1.0000	0.2163	0.1592	0.1627
PAYAMT4	0.1996	0.1801	0.2163	1.0000	0.1518	0.1578
PAYAMT5	0.1485	0.1809	0.1592	0.1518	1.0000	0.1549
PAYAMT6	0.1857	0.1576	0.1627	0.1578	0.1549	1.0000

Table D.5: Variance-covariance matrix of the credit data set: Variables eighteen to twenty-three.

D.4 Principal Component Analysis Results

The results of the PCA of the credit data in chapter 5 is as follows

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
LIMITBAL	0.0685	0.3163	−0.0175	−0.0700	−0.1495	−0.3757	0.1061	−0.0593	0.0215	−0.0175	0.0085
SEX	−0.0217	0.0306	−0.0240	0.0782	0.8841	−0.3954	0.0651	−0.0277	0.0190	−0.0243	−0.0092
EDUCATION	0.0192	−0.0915	0.3164	−0.2466	0.3649	0.5653	−0.3998	0.0809	−0.1545	−0.0629	0.0004
MARRIAGE	−0.0056	−0.0433	−0.4532	0.4446	−0.0533	0.2795	−0.1598	0.0263	−0.0557	−0.0386	−0.0113
AGE	0.0142	0.0666	0.4623	−0.4628	−0.1639	−0.0890	0.0813	−0.0129	0.0468	−0.0079	−0.0212
PAY0	0.1611	−0.2981	0.0175	−0.0160	−0.0345	−0.0399	−0.0866	0.0214	−0.0304	−0.0030	0.0466
PAY2	0.1941	−0.3348	−0.0178	−0.0499	−0.0370	−0.0611	−0.0920	0.0156	−0.0334	−0.0063	0.0293
PAY3	0.2002	−0.3439	−0.0617	−0.0843	−0.0237	−0.0701	−0.0209	−0.0265	−0.0114	0.0249	−0.1065
PAY4	0.2070	−0.3446	−0.0889	−0.1094	−0.0078	−0.0872	0.0392	−0.0562	0.0443	0.0568	−0.0077
PAY5	0.2110	−0.3316	−0.1087	−0.1156	0.0017	−0.0869	0.1038	−0.0339	0.0543	−0.0670	0.0095
PAY6	0.2064	−0.3058	−0.1109	−0.1099	0.0123	−0.0911	0.1388	0.0469	−0.0009	−0.0144	0.0223
BILLAMT1	0.3344	0.1398	0.1509	0.1680	−0.0132	0.0176	−0.0655	−0.0228	−0.0646	−0.0115	0.0103
BILLAMT2	0.3458	0.1402	0.1271	0.1493	0.0046	0.0551	−0.0145	−0.0760	−0.0457	0.0369	−0.1327
BILLAMT3	0.3498	0.1462	0.0839	0.1048	0.0299	0.0854	−0.0034	−0.1259	0.0797	0.1222	0.1000
BILLAMT4	0.3539	0.1454	0.0705	0.1006	0.0226	0.0642	0.0659	−0.0369	0.0267	−0.1235	0.0340
BILLAMT5	0.3518	0.1438	0.0566	0.0887	0.0223	0.0250	0.0802	0.1131	−0.0694	0.0055	0.0468
BILLAMT6	0.3445	0.1419	0.0475	0.0841	0.0105	−0.0215	−0.0496	0.1633	0.0808	−0.0138	−0.0061
PAYAMT1	0.0942	0.1491	−0.2819	−0.2804	0.0927	0.2385	0.2029	−0.2672	0.0363	0.2363	−0.7283
PAYAMT2	0.0787	0.1399	−0.3018	−0.3110	0.1079	0.2559	0.1041	−0.2191	0.3934	0.2790	0.5972
PAYAMT3	0.0864	0.1516	−0.2803	−0.2782	0.0362	0.1429	0.2272	0.2302	−0.0336	−0.7847	0.0252
PAYAMT4	0.0778	0.1414	−0.2374	−0.2365	0.0139	−0.0602	0.0240	0.6466	−0.4458	0.4353	0.0738
PAYAMT5	0.0754	0.1372	−0.1943	−0.1762	−0.0568	−0.2532	−0.7017	0.1972	0.4752	−0.0562	−0.1845
PAYAMT6	0.0699	0.1294	−0.2093	−0.2072	−0.0517	−0.1920	−0.3608	−0.5367	−0.5971	−0.1048	0.1610

Table D.6: PCA results of the credit data set: the first eleven principal components.

	PC12	PC13	PC14	PC15	PC16	PC17
LIMITBAL	−0.0991	0.3592	−0.7541	0.0239	−0.0426	0.0249
SEX	0.0561	0.1759	0.1130	−0.0247	0.0002	−0.0212
EDUCATION	−0.1850	0.0851	−0.3726	0.0361	−0.0096	0.0406
MARRIAGE	−0.1411	0.6671	0.1335	−0.0056	−0.0066	0.0143
AGE	−0.0472	0.5774	0.4324	−0.0235	−0.0014	−0.0006
PAY0	0.6242	0.1557	−0.0987	0.6072	0.2425	−0.0161
PAY2	0.3765	0.0810	−0.0838	−0.3216	−0.5588	0.0710
PAY3	0.0994	0.0519	−0.0852	−0.5417	0.1065	0.1572
PAY4	−0.2116	0.0182	−0.0601	−0.1709	0.5853	−0.0760
PAY5	−0.3584	−0.0232	−0.0200	0.1791	0.0772	−0.1451
PAY6	−0.4000	−0.0401	0.0406	0.3622	−0.5012	−0.0543
BILLAMT1	0.0431	−0.0081	0.0134	−0.0753	−0.0486	−0.5491
BILLAMT2	0.0410	−0.0165	0.0282	−0.0528	−0.0252	−0.3708
BILLAMT3	0.0128	−0.0196	0.0407	−0.0354	0.0118	−0.1115
BILLAMT4	−0.0133	−0.0367	0.0488	0.0097	0.0534	0.2012
BILLAMT5	−0.0419	−0.0373	0.0598	0.0557	0.0362	0.4096
BILLAMT6	−0.0590	−0.0490	0.0751	0.0751	0.0179	0.4791
PAYAMT1	0.1116	0.0004	−0.0064	0.0884	−0.0507	0.0468
PAYAMT2	0.1018	0.0299	−0.0062	−0.0417	−0.0339	−0.0317
PAYAMT3	0.1417	−0.0250	0.0196	−0.0806	0.0461	−0.1290
PAYAMT4	0.0252	0.0012	0.0419	−0.0398	0.0486	−0.1242
PAYAMT5	−0.0417	−0.0720	0.0917	0.0329	−0.0087	−0.0663
PAYAMT6	−0.0428	−0.0762	0.1426	0.0398	0.0077	0.0918

Table D.7: PCA results of the credit data set: variables twelve to seventeen.

	PC18	PC19	PC20	PC21	PC22	PC23
LIMITBAL	−0.0062	0.0070	−0.0099	0.0145	−0.0007	0.0028
SEX	0.0037	−0.0020	0.0018	−0.0007	−0.0008	0.0009
EDUCATION	−0.0043	0.0022	−0.0027	0.0006	0.0014	0.0017
MARRIAGE	0.0056	−0.0007	−0.0025	−0.0002	0.0007	−0.0012
AGE	0.0056	−0.0058	0.0000	−0.0016	0.0000	0.0003
PAY0	−0.1351	−0.0329	−0.0051	−0.0003	−0.0028	0.0002
PAY2	0.4850	0.1387	−0.0311	0.0127	0.0032	0.0000
PAY3	−0.6067	−0.3050	0.0253	−0.0280	−0.0085	−0.0048
PAY4	0.2285	0.5673	0.0192	0.0018	0.0140	−0.0014
PAY5	0.3847	−0.6749	0.0272	0.0234	−0.0082	−0.0006
PAY6	−0.3952	0.3194	−0.0434	−0.0126	0.0001	0.0046
BILLAMT1	−0.0501	0.0129	0.4148	−0.4325	−0.1839	−0.3175
BILLAMT2	−0.0515	−0.0012	0.0399	0.3438	0.3292	0.6464
BILLAMT3	−0.0392	−0.0179	−0.4835	0.4970	−0.0857	−0.5262
BILLAMT4	0.0410	−0.0089	−0.5215	−0.4885	−0.3632	0.3457
BILLAMT5	0.0571	−0.0163	0.0683	−0.2500	0.7185	−0.2265
BILLAMT6	0.0468	0.0248	0.5139	0.3383	−0.4265	0.0715
PAYAMT1	0.0650	0.0186	0.0435	−0.0673	−0.0446	−0.0846
PAYAMT2	−0.0460	−0.0219	0.1466	−0.0707	0.0380	0.1244
PAYAMT3	−0.0281	0.0507	0.0012	0.1235	0.0264	−0.0631
PAYAMT4	0.0236	−0.0484	−0.1118	0.0026	−0.0812	0.0421
PAYAMT5	−0.0168	0.0002	−0.1004	−0.0695	0.0949	−0.0082
PAYAMT6	−0.0053	0.0004	0.0347	0.0274	−0.0172	0.0083

Table D.8: PCA results of the credit data set: variables eighteen to twenty-three.

D.5 Neural Network: 7 Fold Cross Validation

The output of the 7 fold cross validation for the neural network discussed in chapter 5 is as follows:

```
################################
## Output of repeated 7 fold cross validation for neural network.
## using original data
################################

Neural Network

30000 samples
23 predictor
2 classes: 'no', 'yes'

Pre-processing: centered (23), scaled (23)
Resampling: Cross-Validated (7 fold, repeated 10 times)
Summary of sample sizes: 25714, 25715, 25715, 25714, 25714,
   25714, ...
Resampling results across tuning parameters:

  size  decay  ROC        Sens       Spec
3     0.1    0.7688217  0.9490967  0.3600663
3     0.2    0.7682074  0.9497260  0.3590416
3     0.3    0.7685065  0.9497859  0.3582731
3     0.4    0.7688832  0.9503295  0.3554702
3     0.5    0.7695097  0.9499828  0.3567209
4     0.1    0.7719541  0.9502182  0.3579265
4     0.2    0.7721799  0.9506334  0.3575799
4     0.3    0.7716161  0.9507532  0.3561031
4     0.4    0.7715486  0.9513738  0.3545811
4     0.5    0.7720581  0.9516862  0.3541893
5     0.1    0.7723060  0.9501840  0.3570374
5     0.2    0.7730059  0.9502353  0.3578662
5     0.3    0.7737250  0.9504665  0.3562839
5     0.4    0.7729950  0.9516519  0.3541742
5     0.5    0.7733563  0.9513481  0.3555304
6     0.1    0.7734534  0.9496704  0.3611061
6     0.2    0.7738877  0.9502567  0.3574593
6     0.3    0.7740014  0.9508646  0.3579114
6     0.4    0.7735487  0.9512411  0.3559675
6     0.5    0.7737844  0.9515022  0.3563593
7     0.1    0.7734280  0.9501027  0.3585594
7     0.2    0.7739231  0.9497302  0.3579265
7     0.3    0.7740986  0.9507661  0.3583484
7     0.4    0.7739375  0.9512625  0.3564045
```

```
7     0.5    0.7740701  0.9508730  0.3556359

ROC was used to select the optimal model using the largest value.
The final values used for the model were size = 7 and decay =
   0.3.

################################
################################
```

Listing D.1: R Script: 7-fold NN Cross Validation: Credit Data

D.6 Support Vector Machine: 7 Fold Cross Validation

The output of the 7 fold cross validation for the neural network discussed in chapter 5 is as follows

```
################################
## Output of repeated 7 fold cross validation for support vector
   machine.
## using original data
################################

Support Vector Machines with Radial Basis Function Kernel

30000 samples
23 predictor
2 classes: 'no', 'yes'

Pre-processing: centered (23), scaled (23)
Resampling: Cross-Validated (7 fold, repeated 10 times)
Summary of sample sizes: 25714, 25715, 25715, 25714, 25714,
   25714, ...
Resampling results across tuning parameters:

  C     sigma  ROC        Sens       Spec
0.10  0.01   0.7137509  0.9644188  0.2610882
0.10  0.03   0.7118717  0.9613359  0.2990398
0.10  0.05   0.7151410  0.9605652  0.3132144
0.25  0.01   0.7117412  0.9642903  0.2684042
0.25  0.03   0.7130931  0.9608221  0.3086420
0.25  0.05   0.7187386  0.9609505  0.3109282
0.50  0.01   0.7079385  0.9626204  0.2839506
0.50  0.03   0.7160088  0.9609505  0.3104710
0.50  0.05   0.7197640  0.9601798  0.3127572
0.75  0.01   0.7072279  0.9624920  0.2894376
0.75  0.03   0.7174663  0.9597945  0.3118427
```

```
0.75   0.05    0.7217062   0.9606936   0.3155007
1.00   0.01    0.7070711   0.9623635   0.2949246
1.00   0.03    0.7192409   0.9597945   0.3123000
1.00   0.05    0.7213817   0.9603083   0.3155007

ROC was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.05 and C =
    0.75.

################################
################################
```

Listing D.2: Output of the 7-fold NN Cross Validation: Credit Data

D.7 R Code for the Credit Data Set Neural Network

The R script for the neural network model built for the credit data set example in chapter 5 is as follows

```
rm(list=ls())
library(caret)

#####################################
# read in data
#####################################
mydata <- read.csv("default of credit card clients.csv", sep = ";")
str(mydata)
#data imported correctly

#remove ID variable
mydata <- mydata[,-1]
#head(mydata)

#check for any missing variables
apply(mydata,2,function(x) sum(is.na(x)))

#change dependent variable name
names(mydata)[names(mydata) == "default.payment.next.month"] <- "
    defaultPayment"

#change dependent variable coding
indexes <- which(mydata$defaultPayment == 1)
mydata$defaultPayment[indexes] <- 'yes'
mydata$defaultPayment[-indexes] <- 'no'

#changing response variable to factor
```

```
mydata$defaultPayment <- factor(mydata$defaultPayment, levels = c('no', 'yes')
    )

#distribution of classes
cbind(freq = table(mydata$defaultPayment), percentage = prop.table(table(
    mydata$defaultPayment))*100)

#have a look at the first ten observation
head(mydata, n=10)

# output to latex for thesis:
library(Hmisc)
latex(head(mydata, n=10), file="defaultPayment10obs.tex")

# simple summary statistics and output to latex for thesis:
summary(mydata)
latex(summary(mydata), file="defaultPaymentsummarystats.tex")

# variance covariance matrix and output to latex for thesis:
var.cor <- cor(mydata[,-24])
latex(round(var.cor,4), file="defaultPaymentcorrelations.tex")

library(GGally)
# diagram of covariance
ggcorr(mydata[,-24])

#drawing box plots of the feature variables
boxplot(mydata[,-c(2,3,4,6,7,8,9,10,11,24)])

#checking for outliers
outvals <- boxplot(mydata[,-c(2,3,4,6,7,8,9,10,11,24)], plot = FALSE)$out

#diagram to display distribution of classes with respect to given variables
with(mydata, qplot(AGE, LIMIT_BAL, colour=defaultPayment, cex=2, ylab = 'LIMIT
    BALANCE'))

# Class distribution: scatterplot
plot(mydata[,-24], col=c("red","blue")[mydata$defaultPayment])
dev.print(device=postscript, file="CreditScatterplotClass.eps")


# train/test split
# indexes <- createDataPartition(mydata$defaultPayment, times = 1, p = 0.7,
    list = FALSE)
# save(indexes, file="E://Project//Rscript//indexes70.Rdata")

load("indexes70.Rdata")
```

```
train <- mydata[indexes,]
test <- mydata[-indexes,]

ctrl.1 <- trainControl(method = "none")

#####################################
#training neural networks
#five models with between 3-7 nodes in the hidden layer
#####################################

nn3nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'Accuracy', tuneGrid = expand.grid(size = 3, decay = 0.5), trace
    = FALSE, preProcess = c("center", "scale"), trControl=ctrl.1, maxit = 100)

preds <- predict(nn3nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

#####################################
#####################################

nn4nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'Accuracy', tuneGrid = expand.grid(size = 4, decay = 0.5), trace
    = FALSE, preProcess = c("center", "scale"), trControl=ctrl.1, maxit = 100)

preds <- predict(nn4nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

#####################################
#####################################

nn5nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'Accuracy', tuneGrid = expand.grid(size = 5, decay = 0.5), trace
    = FALSE, preProcess = c("center", "scale"), trControl=ctrl.1, maxit = 100)

preds <- predict(nn5nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

#####################################
#####################################

nn6nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'Accuracy', tuneGrid = expand.grid(size = 6, decay = 0.5), trace
    = FALSE, preProcess = c("center", "scale"), trControl=ctrl.1, maxit = 100)

preds <- predict(nn6nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

#####################################
```

```
####################################

nn7nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'Accuracy', tuneGrid = expand.grid(size = 7, decay = 0.5), trace
    = FALSE, preProcess = c("center", "scale"), trControl=ctrl.1, maxit = 100)

preds <- predict(nn7nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

####################################
####################################
```

Listing D.3: NN Credit Data: 70/30

```
rm(list=ls())
library(caret)

####################################
# read in data
####################################
# Linux:
# mydata <- read.csv("../../default of credit card clients.csv", sep = ";")

mydata <- read.csv("default of credit card clients.csv", sep = ";")
str(mydata)
#data imported correctly

#remove ID variable
mydata <- mydata[,-1]
#head(mydata)

#change dependent variable name
names(mydata)[names(mydata) == "default.payment.next.month"] <- "
    defaultPayment"
#change dependent variable coding
indexes <- which(mydata$defaultPayment == 1)
mydata$defaultPayment[indexes] <- 'yes'
mydata$defaultPayment[-indexes] <- 'no'

#changing response variable to factor
mydata$defaultPayment <- factor(mydata$defaultPayment, levels = c('no', 'yes')
    )

# train/test split
# indexes <- createDataPartition(mydata$defaultPayment, times = 1, p = 0.8,
    list = FALSE)
# save(indexes, file="E://Project//Rscript//indexes80.Rdata")

load("indexes80.Rdata")
```

```
train <- mydata[indexes,]
test <- mydata[-indexes,]

ctrl.1 <- trainControl(method = "none")

#####################################
#training neural networks
#five models with between 3-7 nodes in the hidden layer
#####################################

nn3nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'Accuracy', tuneGrid = expand.grid(size = 3, decay = 0.5), trace
    = FALSE, preProcess = c("center", "scale"), trControl=ctrl.1, maxit = 100)

preds <- predict(nn3nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

#####################################
#####################################

nn4nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'Accuracy', tuneGrid = expand.grid(size = 4, decay = 0.5), trace
    = FALSE, preProcess = c("center", "scale"), trControl=ctrl.1, maxit = 100)

preds <- predict(nn4nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

#####################################
#####################################

nn5nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'Accuracy', tuneGrid = expand.grid(size = 5, decay = 0.5), trace
    = FALSE, preProcess = c("center", "scale"), trControl=ctrl.1, maxit = 100)

preds <- predict(nn5nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

#####################################
#####################################

nn6nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'Accuracy', tuneGrid = expand.grid(size = 6, decay = 0.5), trace
    = FALSE, preProcess = c("center", "scale"), trControl=ctrl.1, maxit = 100)

preds <- predict(nn6nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)
```

```
#####################################
#####################################

nn7nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'Accuracy', tuneGrid = expand.grid(size = 7, decay = 0.5), trace
    = FALSE, preProcess = c("center", "scale"), trControl=ctrl.1, maxit = 100)

preds <- predict(nn7nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

#####################################
#####################################
```

Listing D.4: NN Credit Data: 80/20

```
rm(list=ls())
library(caret)

#####################################
# read in data
#####################################
# Linux:
# mydata <- read.csv("../../default of credit card clients.csv", sep = ";")

mydata <- read.csv("default of credit card clients.csv", sep = ";")
str(mydata)
#data imported correctly

#remove ID variable
mydata <- mydata[,-1]
#head(mydata)

#change dependent variable name
names(mydata)[names(mydata) == "default.payment.next.month"] <- "
    defaultPayment"
#change dependent variable coding
indexes <- which(mydata$defaultPayment == 1)
mydata$defaultPayment[indexes] <- 'yes'
mydata$defaultPayment[-indexes] <- 'no'

#changing response variable to factor
mydata$defaultPayment <- factor(mydata$defaultPayment, levels = c('no', 'yes')
    )

# train/test split
# indexes <- createDataPartition(mydata$defaultPayment, times = 1, p = 0.9,
    list = FALSE)
# save(indexes, file="E://Project//Rscript//indexes90.Rdata")
```

```
load("indexes90.Rdata")

train <- mydata[indexes,]
test <- mydata[-indexes,]

#create train control to set up ROC as the performance measure
ctrl.1 <- trainControl(method = "none")

library(pROC)

#####################################
#training neural networks
#five models with between 3-7 nodes in the hidden layer
#####################################

nn3nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
    metric = 'ROC', tuneGrid = expand.grid(size = 3, decay = 0.5), trace =
   FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

confusionMatrix(predict(nn3nodes, newdata=train[,-24]), train[,24])
confusionMatrix(predict(nn3nodes, newdata=test[,-24]), test[,24])

modpred<- predict(nn3nodes, newdata=test[,1:23], type="prob")

nn3nodesROC <- roc(predictor=modpred$no, response=test$defaultPayment, levels=
   rev(levels(test$defaultPayment)))

nn3nodesROC$auc

#####################################
#####################################

nn4nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
    metric = 'ROC', tuneGrid = expand.grid(size = 4, decay = 0.5), trace =
   FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

confusionMatrix(predict(nn4nodes, newdata=train[,-24]), train[,24])
confusionMatrix(predict(nn4nodes, newdata=test[,-24]), test[,24])

modpred<- predict(nn4nodes, newdata=test[,1:23], type="prob")

nn4nodesROC <- roc(predictor=modpred$no, response=test$defaultPayment, levels=
   rev(levels(test$defaultPayment)))

nn4nodesROC$auc

#####################################
#####################################
```

```
nn5nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
    metric = 'ROC', tuneGrid = expand.grid(size = 5, decay = 0.5), trace =
   FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

confusionMatrix(predict(nn5nodes, newdata=train[,-24]), train[,24])
confusionMatrix(predict(nn5nodes, newdata=test[,-24]), test[,24])

modpred<- predict(nn5nodes, newdata=test[,1:23], type="prob")

nn5nodesROC <- roc(predictor=modpred$no, response=test$defaultPayment, levels=
   rev(levels(test$defaultPayment)))

nn5nodesROC$auc

#######################################
#######################################

nn6nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
    metric = 'ROC', tuneGrid = expand.grid(size = 6, decay = 0.5), trace =
   FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

confusionMatrix(predict(nn6nodes, newdata=train[,-24]), train[,24])
confusionMatrix(predict(nn6nodes, newdata=test[,-24]), test[,24])

modpred<- predict(nn6nodes, newdata=test[,1:23], type="prob")

nn6nodesROC <- roc(predictor=modpred$no, response=test$defaultPayment, levels=
   rev(levels(test$defaultPayment)))

nn6nodesROC$auc

#######################################
#######################################

nn7nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
    metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.5), trace =
   FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

confusionMatrix(predict(nn7nodes, newdata=train[,-24]), train[,24])
confusionMatrix(predict(nn7nodes, newdata=test[,-24]), test[,24])

modpred<- predict(nn7nodes, newdata=test[,1:23], type="prob")

nn7nodesROC <- roc(predictor=modpred$no, response=test$defaultPayment, levels=
   rev(levels(test$defaultPayment)))

nn7nodesROC$auc
```

```
###################################
###################################

# plot ROC curves
plot(nn3nodesROC, col = "green", main = "ROC curve of NN with different number
    of nodes")

# Draw a legend.
legend(0.6, 0.35, c('3', '4','5', '6', '7'), c('green', 'red', 'blue', 'black'
   , 'yellow'))

lines(nn4nodesROC, col = "red")
lines(nn5nodesROC, col = "blue")
lines(nn6nodesROC, col = "black")
lines(nn7nodesROC, col = "yellow")

#####################################
#####################################

accuracyRate <- 1:6
AUC <- 1:6
annNodes <- data.frame(accuracyRate, AUC)

ctrl.1 <- trainControl(method = "none", summaryFunction=twoClassSummary,
    classProbs=TRUE)

#####################################
#####################################

nn8nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet',
     metric = 'ROC', tuneGrid = expand.grid(size = 8, decay = 0.5), trace =
    FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

preds <- predict(nn8nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

annNodes$accuracyRate[1] <- confusionMatrix(preds, test$defaultPayment)$
    overall[1]

#####################################
#####################################

nn10nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet'
    , metric = 'ROC', tuneGrid = expand.grid(size = 10, decay = 0.5), trace =
    FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

preds <- predict(nn10nodes, test[,1:23])
```

```
confusionMatrix(preds, test$defaultPayment)

annNodes$accuracyRate[2] <- confusionMatrix(preds, test$defaultPayment)$
    overall[1]

#####################################
#####################################

nn15nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet'
    , metric = 'ROC', tuneGrid = expand.grid(size = 15, decay = 0.5), trace =
    FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

preds <- predict(nn15nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

annNodes$accuracyRate[3] <- confusionMatrix(preds, test$defaultPayment)$
    overall[1]

#####################################
#####################################

nn20nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet'
    , metric = 'ROC', tuneGrid = expand.grid(size = 20, decay = 0.5), trace =
    FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

preds <- predict(nn20nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

annNodes$accuracyRate[4] <- confusionMatrix(preds, test$defaultPayment)$
    overall[1]

#####################################
#####################################

nn30nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet'
    , metric = 'ROC', tuneGrid = expand.grid(size = 30, decay = 0.5), trace =
    FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

preds <- predict(nn30nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

annNodes$accuracyRate[5] <- confusionMatrix(preds, test$defaultPayment)$
    overall[1]

#####################################
#####################################
```

```
nn35nodes <- train(x = train[,1:23], y = train$defaultPayment, method = 'nnet'
    , metric = 'ROC', tuneGrid = expand.grid(size = 35, decay = 0.5), trace =
    FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

preds <- predict(nn35nodes, test[,1:23])
confusionMatrix(preds, test$defaultPayment)

annNodes$accuracyRate[6] <- confusionMatrix(preds, test$defaultPayment)$
    overall[1]

#######################################
#######################################

numPC <- c(8,10,15,20,30,35)
plot(numPC, annNodes$accuracyRate, xlab = 'Number of nodes', ylab = 'Accuracy
    rate')
lines(numPC, annNodes$accuracyRate)

#######################################
#######################################
```

Listing D.5: NN Credit Data: 90/10

D.8 R Code for the Credit Data Set Support Vector Machine

The R script for the support vector machine model built for the credit data set example in chapter 5 is as follows

```
rm(list=ls())
library(caret)
library(kernlab)
library(pROC)

#######################################
# read in data
#######################################
# Linux:
# mydata <- read.csv("../../default of credit card clients.csv", sep = ";")

mydata <- read.csv("default of credit card clients.csv", sep = ";")
str(mydata)
#data imported correctly

#remove ID variable
mydata <- mydata[,-1]
```

```
#head(mydata)

#change dependent variable name
names(mydata)[names(mydata) == "default.payment.next.month"] <- "
    defaultPayment"
#change dependent variable coding
indexes <- which(mydata$defaultPayment == 1)
mydata$defaultPayment[indexes] <- 'yes'
mydata$defaultPayment[-indexes] <- 'no'

#changing response variable to factor
mydata$defaultPayment <- factor(mydata$defaultPayment, levels = c('no', 'yes')
    )

#####################################
# Use the same train and test data used in NN (90&10 partition)
# Load the data set
#####################################

load("indexes90.Rdata")

train <- mydata[indexes,]
test <- mydata[-indexes,]

#create a small train set
indexessmall <- createDataPartition(mydata$defaultPayment, times = 1, p =
    0.15, list = FALSE)
trainsmall <- mydata[indexessmall,]

#####################################
#####################################

#create train control to set up ROC as the performance measure
ctrl.1 <- trainControl(summaryFunction=twoClassSummary, classProbs=TRUE)
ctrl.2 <- trainControl(method = "none", summaryFunction=twoClassSummary,
    classProbs=TRUE)

#####################################
#####################################

#set up tune grid for linear svm
svmGrid <- expand.grid(C = c(0.1, 0.25, 0.5, 0.75, 1))

# check code runs on smaller subset
svmLinsmall <- train(x = trainsmall[,1:23], y = trainsmall$defaultPayment,
    method = 'svmLinear', metric = 'Accuracy', preProcess = c("center", "scale"
    ), tuneGrid = svmGrid, trControl = ctrl.1)
```

```
#extract parameters that maximise accuracy
svmGrid <- expand.grid(C = 1)

# Run SVM on full data set
# Very slow
svmLin <- train(x = train[,1:23], y = train$defaultPayment, method = '
    svmLinear', metric = 'ROC', preProcess = c("center", "scale"), tuneGrid =
    svmGrid, trControl = ctrl.2)

confusionMatrix(predict(svmLin, newdata=train[,-24]), train[,24])
confusionMatrix(predict(svmLin, newdata=test[,-24]), test[,24])

modpred<- predict(svmLin, newdata=test[,1:23], type="prob")

svmLinROC <- roc(predictor=modpred$no, response=test$defaultPayment, levels=
    rev(levels(test$defaultPayment)))

svmLinROC$auc

#####################################
#####################################

#set up tune grid
svmGrid <- expand.grid(C = c(0.1, 0.25, 0.5, 0.75, 1), sigma = c(0.01, 0.03,
    0.05))

# check code runs on smaller subset
svmRadsmall <- train(x = trainsmall[,1:23], y = trainsmall$defaultPayment,
    method = 'svmRadial', metric = 'Accuracy', preProcess = c("center", "scale"
    ), tuneGrid = svmGrid, trControl = ctrl.1)

#extract parameters that maximise accuracy
svmGrid <- expand.grid(C = 0.75, sigma = 0.05)

# Run SVM on full data set
# Very slow
svmRad <- train(x = train[,1:23], y = train$defaultPayment, method = '
    svmRadial', metric = 'ROC', preProcess = c("center", "scale"), tuneGrid =
    svmGrid, trControl = ctrl.2)

confusionMatrix(predict(svmRad, newdata=train[,-24]), train[,24])
confusionMatrix(predict(svmRad, newdata=test[,-24]), test[,24])

modpred<- predict(svmRad, newdata=test[,1:23], type="prob")

svmRadROC <- roc(predictor=modpred$no, response=test$defaultPayment, levels=
    rev(levels(test$defaultPayment)))
```

```
svmRadROC$auc

###################################
###################################

#set up tune grid
svmGrid <- expand.grid(degree = c(1, 2, 3), scale = c(0.2, 0.4, 0.6, 0.8, 1),
    C = 1)

# check code runs on smaller subset
svmPolysmall <- train(x = trainsmall[,1:23], y = trainsmall$defaultPayment,
    method = 'svmPoly', metric = 'Accuracy', preProcess = c("center", "scale"),
     tuneGrid = svmGrid, trControl = ctrl.1)


#extract parameters that maximise accuracy
svmGrid <- expand.grid(degree = 1, scale = 0.4, C = 1)

# Run SVM on full data set
# Very slow
svmPoly <- train(x = train[,1:23], y = train$defaultPayment, method = 'svmPoly
    ', metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.2)

confusionMatrix(predict(svmPoly, newdata=train[,-24]), train[,24])
confusionMatrix(predict(svmPoly, newdata=test[,-24]), test[,24])

modpred<- predict(svmPoly, newdata=test[,1:23], type="prob")

svmPolyROC <- roc(predictor=modpred$no, response=test$defaultPayment, levels=
    rev(levels(test$defaultPayment)))

svmPolyROC$auc

###################################
###################################

# plot ROC curves
plot(svmLinROC, col = "green", main = "ROC curve of SVM with different kernels
    ")
# Draw a legend.
legend(0.4, 0.35, c('Linear', 'Radial','Polynomial'), c('green', 'red', 'blue'
    ))

lines(svmRadROC, col = "red")
lines(svmPolyROC, col = "blue")

###################################
```

```
####################################
```

Listing D.6: R Script: SWM: Credit Data

D.9 R Code for the Credit Data Set 7 Fold Cross Validation

The R script for the neural network and support vector machine models built for the credit data set example with 7 fold cross validation in chapter 5 is as follows

```
#cross validation
rm(list=ls())
library(doSNOW)
library(caret)

#####################################
# read in data
#####################################
# Linux:
# mydata <- read.csv("../../default of credit card clients.csv", sep = ";")

mydata <- read.csv("default of credit card clients.csv", sep = ";")
str(mydata)
#data imported correctly

#remove ID variable
mydata <- mydata[,-1]
#head(mydata)

#change dependent variable name
names(mydata)[names(mydata) == "default.payment.next.month"] <- "
    defaultPayment"
#change dependent variable coding
indexes <- which(mydata$defaultPayment == 1)
mydata$defaultPayment[indexes] <- 'yes'
mydata$defaultPayment[-indexes] <- 'no'

#changing response variable to factor
mydata$defaultPayment <- factor(mydata$defaultPayment, levels = c('no', 'yes')
    )

# set up train control to do cross-validation
ctrl.1 <- trainControl(method = 'repeatedcv', number = 7, repeats = 10)
ctrl.2 <- trainControl(method = 'repeatedcv', number = 7, repeats = 10,
    classProbs = TRUE, summaryFunction = twoClassSummary)
```

```
###################################
###################################

cl <- makeCluster(6, type = 'SOCK')
registerDoSNOW(cl)

nnetGrid <-  expand.grid(size = seq(from = 3, to = 7, by = 1), decay = seq(
    from = 0.1, to = 0.5, by = 0.1))

svmGrid <- expand.grid(C = c(0.1, 0.25, 0.5, 0.75, 1), sigma = c(0.01, 0.03,
    0.05))

NNcv <- train(defaultPayment ~ ., data=mydata, method = 'nnet', metric = 'ROC'
    , trControl = ctrl.2, tuneGrid = nnetGrid, trace = FALSE, preProcess = c("
    center", "scale"), linear.output = FALSE)


# check code runs on smaller subset
SVMcv <- train(x =mydata[1:1000,1:23], y = mydata$defaultPayment[1:1000],
    method = "svmRadial", metric = 'ROC', tuneGrid = svmGrid, trControl=ctrl.2,
     preProcess = c("center", "scale"))

# Run SVM on full data set
# Very slow
SVMcv <- train(defaultPayment ~ ., data=mydata, method = "svmRadial", metric =
     'ROC', tuneGrid = svmGrid, trControl=ctrl.2, preProcess = c("center", "
    scale"))

stopCluster(cl)

###################################
###################################

NNcv$results[3]
SVMcv$results[3]

###################################
###################################

#import the function from Github to plot NN model
library(devtools)
source_url('https://gist.githubusercontent.com/Peque/41a9e20d6687f2f3108d/raw/
    85e14f3a292e126f1454864427e3a189c2fe33f3/nnet_plot_update.r')

#plot cross validation results
plot(NNcv)
plot(SVMcv)
```

```
#plot models
plot.nnet(NNcv)
plor.svm(SVMcv)

#####################################
#####################################
```

Listing D.7: R Script: 7-fold SVM Cross Validation: Credit Data

D.10 R Code for the Dimension Reduction Techniques

The R script for the neural network and support vector machine models built using the dimensionally reduced feature space in chapter 5 is as follows

```
rm(list=ls())
library(caret)

#####################################
# read in data
#####################################
# Linux:
# mydata <- read.csv("../../default of credit card clients.csv", sep = ";")

mydata <- read.csv("default of credit card clients.csv", sep = ";")
str(mydata)
#data imported correctly

#remove ID variable
mydata <- mydata[,-1]
#head(mydata)

#change dependent variable name
names(mydata)[names(mydata) == "default.payment.next.month"] <- "
    defaultPayment"
#change dependent variable coding
indexes <- which(mydata$defaultPayment == 1)
mydata$defaultPayment[indexes] <- 'yes'
mydata$defaultPayment[-indexes] <- 'no'

#changing response variable to factor
mydata$defaultPayment <- factor(mydata$defaultPayment, levels = c('no', 'yes')
    )

load("indexes90.Rdata")

#####################################
# Conduct PCA
```

```
# Fit model
#####################################

prin_comp <- prcomp(mydata[,-24], scale. = T)

# Interpretation
prin_comp
library(Hmisc)
latex(round(prin_comp$rotation,4), file="PrinCompScoresCredit.tex")
prin_comp$rotation
# Interpretation: Not overly clear unless you are a wine expert or chemist?
biplot(prin_comp, scale = 0)

# compute standard deviation of each principal component
std_dev <- prin_comp$sdev

# compute variance
pr_var <- std_dev^2

# check variance of first 10 components
pr_var[1:10]

# proportion of variance explained
prop_varex <- pr_var/sum(pr_var)
prop_varex[1:15]
sum(prop_varex[1:15])

# scree plot
plot(prop_varex, xlab = "Principal Component", ylab = "Proportion of Variance
    Explained", type = "b")

# cumulative scree plot
plot(cumsum(prop_varex), xlab = "Principal Component", ylab = "Cumulative
    Proportion of Variance Explained", type = "b")

# add a data set with principal components
mydataPCA <- data.frame(defaultPayment = mydata$defaultPayment, prin_comp$x)

trainPCA <- mydataPCA[indexes,]
testPCA <- mydataPCA[-indexes,]

#we are interested in first 15 PCAs
trainPCA1 <- trainPCA[,1:16]
testPCA1 <- testPCA[,1:16]

#create train control to set up ROC as the performance measure
ctrl.1 <- trainControl(method = "none")
#ctrl.1 <- trainControl(summaryFunction=twoClassSummary,   classProbs=TRUE)
```

```
library(pROC)

#####################################
#####################################

nnPCA <- train(x = trainPCA1[,2:16], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
    = FALSE, preProcess = c("center", "scale"), maxit = 100, trControl = ctrl
    .1)

preds <- predict(nnPCA, testPCA1[,-1])
confusionMatrix(preds, testPCA1$defaultPayment)

modpred<- predict(nnPCA, newdata=testPCA1[,-1], type="prob")

nnPCAROC <- roc(predictor=modpred$no, response=testPCA1$defaultPayment, levels
    =rev(levels(testPCA1$defaultPayment)))

nnPCAROC$auc

#####################################
#####################################

#parameters that maximise accuracy
svmGrid <- expand.grid(C = 0.75, sigma = 0.05)

svmPCA <- train(x = trainPCA1[,2:16], y = trainPCA1$defaultPayment, method = '
    svmRadial', metric = 'ROC', preProcess = c("center", "scale"), tuneGrid =
    svmGrid, trControl = ctrl.1)

preds <- predict(svmPCA, testPCA1[,-1])
confusionMatrix(preds, testPCA1$defaultPayment)

modpred<- predict(svmPCA, newdata=testPCA1[,-1], type="prob")

svmPCAROC <- roc(predictor=modpred$no, response=testPCA1$defaultPayment,
    levels=rev(levels(testPCA1$defaultPayment)))

svmPCAROC$auc

#####################################
#####################################
```

Listing D.8: R Script: Reduced Credit Data

```
rm(list=ls())
library(caret)
```

```
###################################
# read in data
###################################
# Linux:
# mydata <- read.csv("../../default of credit card clients.csv", sep = ";")
mydata <- read.csv("default of credit card clients.csv", sep = ";")
str(mydata)
#data imported correctly

#remove ID variable
mydata <- mydata[,-1]
#head(mydata)

#change dependent variable name
names(mydata)[names(mydata) == "default.payment.next.month"] <- "
    defaultPayment"
#change dependent variable coding
indexes <- which(mydata$defaultPayment == 1)
mydata$defaultPayment[indexes] <- 'yes'
mydata$defaultPayment[-indexes] <- 'no'

#changing response variable to factor
mydata$defaultPayment <- factor(mydata$defaultPayment, levels = c('no', 'yes')
    )

load("indexes90.Rdata")

#using lda method to reduce feature space
library( MASS )
linearDis = lda(defaultPayment ~ ., data = mydata )

# Interpretation
projected_data = as.matrix( mydata[, -24] ) %*% linearDis$scaling
plot( projected_data, col = mydata[,24], pch = 19 )

# add a training set with linear discriminants
mydataLDA <- as.matrix( mydata[, -24] ) %*% linearDis$scaling
mydataLDA <- as.data.frame(mydataLDA)
mydataLDA <- data.frame(mydataLDA, defaultPayment = mydata$defaultPayment)

trainLDA <- mydataLDA[indexes,]
testLDA <- mydataLDA[-indexes,]

#create train control to set up ROC as the performance measure
ctrl.1 <- trainControl(method = "none")
#ctrl.1 <- trainControl(summaryFunction=twoClassSummary,   classProbs=TRUE)

###################################
```

```
#####################################
library(pROC)

nnLDA <- train(defaultPayment ~ ., data = trainLDA, method = 'nnet', metric =
    'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace = FALSE,
    preProcess = c("center", "scale"), maxit = 100, trControl = ctrl.1)

outofsampedata <- as
preds <- predict(nnLDA, newdata=data.frame(LD1=testLDA$LD1))
confusionMatrix(preds, testLDA$defaultPayment)


modpred<- predict(nnLDA, newdata=testLDA[,-2], type="prob")

nnLDAROC <- roc(predictor=modpred$no, response=testLDA$defaultPayment, levels=
    rev(levels(testLDA$defaultPayment)))

nnLDAROC$auc

#####################################
#####################################

#parameters that maximise accuracy
svmGrid <- expand.grid(C = 1, sigma = 0.015)

svmLDA <- train(defaultPayment ~ ., data = trainLDA, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmLDA, testLDA[,-2])
confusionMatrix(preds, testLDA$defaultPayment)

modpred<- predict(svmLDA, newdata=testLDA[,-2], type="prob")

svmLDAROC <- roc(predictor=modpred$no, response=testLDA$defaultPayment, levels
    =rev(levels(testLDA$defaultPayment)))

svmLDAROC$auc

#####################################
#####################################

# plot ROC curves
plot(nn7nodesROC, col = "green", main = "ROC curve of NN classifiers")
# Draw a legend.
legend(0.4, 0.35, c('NN', 'PCA-NN','LDA-NN'), c('green', 'red', 'blue'))

lines(nnPCAROC, col = "red")
```

```
lines(nnLDAROC, col = "blue")

###################################
###################################

# plot ROC curves
plot(svmRadROC, col = "green", main = "ROC curve of SVM classifiers")
# Draw a legend.
legend(0.4, 0.35, c('SVM', 'PCA-SVM','LDA-SVM'), c('green', 'red', 'blue'))

lines(svmPCAROC, col = "red")
lines(svmLDAROC, col = "blue")

###################################
###################################
```

Listing D.9: R Script: Reduced Credit Data

D.11 R Code for the Number of PC Investigation

The R script for the investigation conducted to find out the best number of principal components to use to build the optimal model for the credit data set analysis in chapter 5 is as follows

```
#get principal components
mydataPCA <- data.frame(defaultPayment = mydata$defaultPayment, prin_comp$x)

trainPCA <- mydataPCA[indexes,]
testPCA <- mydataPCA[-indexes,]

#create dataframe for accuracy and AUC
accuracyRate <- 2:23
AUC <- 2:23
annPC <- data.frame(accuracyRate, AUC)

ctrl.1 <- trainControl(method = 'repeatedcv', number = 7, repeats = 10,
    classProbs = TRUE, summaryFunction = twoClassSummary)

###################################
###################################

#we are interested in first 23 PCs
trainPCA1 <- trainPCA[,1:24]
testPCA1 <- testPCA[,1:24]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
```

```
    = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[22] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[22] <- nnModel$results[3]

#######################################
#######################################

#we are interested in first 22 PCs
trainPCA1 <- trainPCA1[,1:23]
testPCA1 <- testPCA1[,1:23]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3),
    trace = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[21] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[21] <- nnModel$results[3]

####################################
####################################

#we are interested in first 21 PCs
trainPCA1 <- trainPCA1[,1:22]
testPCA1 <- testPCA1[,1:22]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3),
    trace = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[20] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]
```

```
## get AUC
annPC$AUC[20] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 20 PCs
trainPCA1 <- trainPCA1[,1:21]
testPCA1 <- testPCA1[,1:21]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3),
    trace = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[19] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[19] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 19 PCs
trainPCA1 <- trainPCA1[,1:20]
testPCA1 <- testPCA1[,1:20]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[18] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[18] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 18 PCs
trainPCA1 <- trainPCA1[,1:19]
```

```
testPCA1 <- testPCA1[,1:19]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[17] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[17] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 17 PCs
trainPCA1 <- trainPCA1[,1:18]
testPCA1 <- testPCA1[,1:18]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3),
    trace = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[16] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[16] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 16 PCs
trainPCA1 <- trainPCA1[,1:17]
testPCA1 <- testPCA1[,1:17]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3),
    trace = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)
```

```
annPC$accuracyRate[15] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[15] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 15 PCs
trainPCA1 <- trainPCA1[,1:16]
testPCA1 <- testPCA1[,1:16]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[14] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[14] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 14 PCs
trainPCA1 <- trainPCA1[,1:15]
testPCA1 <- testPCA1[,1:15]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[13] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[13] <- nnModel$results[3]

#####################################
```

```
###################################

#we are interested in first 13 PCs
trainPCA1 <- trainPCA1[,1:14]
testPCA1 <- testPCA1[,1:14]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[12] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[12] <- nnModel$results[3]

###################################
###################################

#we are interested in first 12 PCs
trainPCA1 <- trainPCA1[,1:13]
testPCA1 <- testPCA1[,1:13]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[11] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[11] <- nnModel$results[3]

###################################
###################################

#we are interested in first 11 PCs
trainPCA1 <- trainPCA1[,1:12]
testPCA1 <- testPCA1[,1:12]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
```

```
    = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[10] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[10] <- nnModel$results[3]

#######################################
#######################################

#we are interested in first 10 PCs
trainPCA1 <- trainPCA1[,1:11]
testPCA1 <- testPCA1[,1:11]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
    = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[9] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[9] <- nnModel$results[3]

####################################
####################################

#we are interested in first 9 PCs
trainPCA1 <- trainPCA1[,1:10]
testPCA1 <- testPCA1[,1:10]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
    = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[8] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]
```

```
## get AUC
annPC$AUC[8] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 8 PCs
trainPCA1 <- trainPCA1[,1:9]
testPCA1 <- testPCA1[,1:9]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[7] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[7] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 7 PCs
trainPCA1 <- trainPCA1[,1:8]
testPCA1 <- testPCA1[,1:8]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[6] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[6] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 6 PCs
trainPCA1 <- trainPCA1[,1:7]
```

```
testPCA1 <- testPCA1[,1:7]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[5] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[5] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 5 PCs
trainPCA1 <- trainPCA1[,1:6]
testPCA1 <- testPCA1[,1:6]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[4] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[4] <- nnModel$results[3]

#####################################
#####################################

#we are interested in first 4 PCs
trainPCA1 <- trainPCA1[,1:5]
testPCA1 <- testPCA1[,1:5]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)
```

```
annPC$accuracyRate[3] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[3] <- nnModel$results[3]

###################################
###################################

#we are interested in first 3 PCs
trainPCA1 <- trainPCA1[,1:4]
testPCA1 <- testPCA1[,1:4]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[2] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[2] <- nnModel$results[3]

###################################
###################################

#we are interested in first 2 PCs
trainPCA1 <- trainPCA1[,1:3]
testPCA1 <- testPCA1[,1:3]

nnModel <- train(x = trainPCA1[,-1], y = trainPCA1$defaultPayment, method = '
    nnet', metric = 'ROC', tuneGrid = expand.grid(size = 7, decay = 0.3), trace
     = FALSE, preProcess = c("center", "scale"), trControl = ctrl.1)

preds <- predict(nnModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

annPC$accuracyRate[1] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
annPC$AUC[1] <- nnModel$results[3]

###################################
```

```
####################################

write.table(annPC, "E:\\annPC.csv", row.names = FALSE, dec = ".", sep = ";")

numPC <- 2:23

plot(numPC, annPC$accuracyRate, xlab = 'Number of Principal Components', ylab
    = 'Accuracy rate')
lines(numPC, annPC$accuracyRate)

plot(numPC, annPC$AUC, xlab = 'Number of Principal Components', ylab = 'AUC')
lines(numPC, annPC$AUC)

####################################
####################################
```

Listing D.10: PCA: NN Credit Data

```
#get principal components
mydataPCA <- data.frame(defaultPayment = mydata$defaultPayment, prin_comp$x)

trainPCA <- mydataPCA[indexes,]
testPCA <- mydataPCA[-indexes,]

#create dataframe for accuracy and AUC
accuracyRate <- 2:23
AUC <- 2:23
svmPC <- data.frame(accuracyRate, AUC)

svmGrid <- expand.grid(C = 0.75, sigma = 0.05)

ctrl.1 <- trainControl(method = 'repeatedcv', number = 7, repeats = 10,
    classProbs = TRUE, summaryFunction = twoClassSummary)

####################################
####################################

#we are interested in first 23 PCs
trainPCA1 <- trainPCA[,1:24]
testPCA1 <- testPCA[,1:24]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)
```

```
svmPC$accuracyRate[22] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[22] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 22 PCs
trainPCA1 <- trainPCA1[,1:23]
testPCA1 <- testPCA1[,1:23]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[21] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[21] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 21 PCs
trainPCA1 <- trainPCA1[,1:22]
testPCA1 <- testPCA1[,1:22]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[20] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[20] <- svmModel$results[3]

#####################################
#####################################
```

```
#we are interested in first 20 PCs
trainPCA1 <- trainPCA1[,1:21]
testPCA1 <- testPCA1[,1:21]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[19] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[19] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 19 PCs
trainPCA1 <- trainPCA1[,1:20]
testPCA1 <- testPCA1[,1:20]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[18] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[18] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 18 PCs
trainPCA1 <- trainPCA1[,1:19]
testPCA1 <- testPCA1[,1:19]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)
```

```
preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[17] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[17] <- svmModel$results[3]

#######################################
#######################################

#we are interested in first 17 PCs
trainPCA1 <- trainPCA1[,1:18]
testPCA1 <- testPCA1[,1:18]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[16] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[16] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 16 PCs
trainPCA1 <- trainPCA1[,1:17]
testPCA1 <- testPCA1[,1:17]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[15] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
```

```
svmPC$AUC[15] <- svmModel$results[3]

###################################
###################################

#we are interested in first 15 PCs
trainPCA1 <- trainPCA1[,1:16]
testPCA1 <- testPCA1[,1:16]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[14] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[14] <- svmModel$results[3]

###################################
###################################

#we are interested in first 14 PCs
trainPCA1 <- trainPCA1[,1:15]
testPCA1 <- testPCA1[,1:15]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[13] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[13] <- svmModel$results[3]

###################################
###################################

#we are interested in first 13 PCs
trainPCA1 <- trainPCA1[,1:14]
testPCA1 <- testPCA1[,1:14]
```

```
svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[12] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[12] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 12 PCs
trainPCA1 <- trainPCA1[,1:13]
testPCA1 <- testPCA1[,1:13]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[11] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[11] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 11 PCs
trainPCA1 <- trainPCA1[,1:12]
testPCA1 <- testPCA1[,1:12]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)
```

```
svmPC$accuracyRate[10] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[10] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 10 PCs
trainPCA1 <- trainPCA1[,1:11]
testPCA1 <- testPCA1[,1:11]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[9] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[9] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 9 PCs
trainPCA1 <- trainPCA1[,1:10]
testPCA1 <- testPCA1[,1:10]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[8] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[8] <- svmModel$results[3]

#####################################
#####################################
```

```
#we are interested in first 8 PCs
trainPCA1 <- trainPCA1[,1:9]
testPCA1 <- testPCA1[,1:9]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[7] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[7] <- svmModel$results[3]

#
    ##############################################################################################


#we are interested in first 7 PCs
trainPCA1 <- trainPCA1[,1:8]
testPCA1 <- testPCA1[,1:8]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[6] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[6] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 6 PCs
trainPCA1 <- trainPCA1[,1:7]
testPCA1 <- testPCA1[,1:7]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
```

```
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[5] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[5] <- svmModel$results[3]

#######################################
#######################################

#we are interested in first 5 PCs
trainPCA1 <- trainPCA1[,1:6]
testPCA1 <- testPCA1[,1:6]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[4] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[4] <- svmModel$results[3]

####################################
####################################

#we are interested in first 4 PCs
trainPCA1 <- trainPCA1[,1:5]
testPCA1 <- testPCA1[,1:5]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[3] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]
```

```
## get AUC
svmPC$AUC[3] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 3 PCs
trainPCA1 <- trainPCA1[,1:4]
testPCA1 <- testPCA1[,1:4]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[2] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[2] <- svmModel$results[3]

#####################################
#####################################

#we are interested in first 2 PCs
trainPCA1 <- trainPCA1[,1:3]
testPCA1 <- testPCA1[,1:3]

svmModel <- train(defaultPayment ~ ., data = trainPCA1, method = 'svmRadial',
    metric = 'ROC', preProcess = c("center", "scale"), tuneGrid = svmGrid,
    trControl = ctrl.1)

preds <- predict(svmModel, testPCA1[-1])
confusionMatrix(preds, testPCA1$defaultPayment)

svmPC$accuracyRate[1] <- confusionMatrix(preds, testPCA1$defaultPayment)$
    overall[1]

## get AUC
svmPC$AUC[1] <- svmModel$results[3]

#####################################
#####################################

write.csv(svmPC, "L:\\svmPC.csv", row.names = FALSE)
```

```
numPC <- 2:23

plot(numPC, svmPC$accuracyRate, xlab = 'Number of Principal Components', ylab
    = 'Accuracy rate')
lines(numPC, svmPC$accuracyRate)

plot(numPC, svmPC$AUC, xlab = 'Number of Principal Components', ylab = 'AUC')
lines(numPC, svmPC$AUC)

########################################
########################################
```

Listing D.11: PCA: SVM Credit Data