A COMPARISON OF SUPPORT VECTOR MACHINES AND TRADITIONAL TECHNIQUES FOR STATISTICAL REGRESSION AND CLASSIFICATION

Trudie Hechter

Thesis presented in partial fulfilment of the requirements for the degree of Master of Commerce at the University of Stellenbosch.

Supervisor: S.J. Steel

April 2004

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

SUMMARY

Since its introduction in Boser et al. (1992), the support vector machine has become a popular tool in a variety of machine learning applications. More recently, the support vector machine has also been receiving increasing attention in the statistical community as a tool for classification and regression. In this thesis support vector machines are compared to more traditional techniques for statistical classification and regression. The techniques are applied to data from a life assurance environment for a binary classification problem and a regression problem. In the classification case the problem is the prediction of policy lapses using a variety of input variables, while in the regression case the goal is to estimate the income of clients from these variables. The performance of the support vector machine is compared to that of discriminant analysis and classification trees in the case of classification, and to that of multiple linear regression and regression trees in regression, and it is found that support vector machines generally perform well compared to the traditional techniques.

OPSOMMING

Sedert die bekendstelling van die ondersteuningspuntalgoritme in Boser et al. (1992), het dit 'n populêre tegniek in 'n verskeidenheid masjienleerteorie applikasies geword. Meer onlangs het die ondersteuningspuntalgoritme ook meer aandag in die statistiese gemeenskap begin geniet as 'n tegniek vir klassifikasie en regressie. In hierdie tesis word ondersteuningspuntalgoritmes vergelyk met meer tradisionele tegnieke vir statistiese klassifikasie en regressie. Die tegnieke word toegepas op data uit 'n lewensversekeringomgewing vir 'n binêre klassifikasie probleem sowel as 'n regressie probleem. In die klassifikasiegeval is die probleem die voorspelling van polisvervallings deur 'n verskeidenheid invoer veranderlikes te gebruik, terwyl in die regressiegeval gepoog word om die inkomste van kliënte met behulp van hierdie veranderlikes te voorspel. Die resultate van die ondersteuningspuntalgoritme word met dié van diskriminant analise en klassifikasiebome vergelyk in die klassifikasiegeval, en met veelvoudige linêere regressie en regressiebome in die regressiegeval. Die gevolgtrekking is dat ondersteuningspuntalgoritmes oor die algemeen goed vaar in vergelyking met die tradisionele tegnieke.

CONTENTS

Chapter 1

Introduction

1. Changes in the statistical environment during the last two decades

Although it is probably presumptuous to try to define the field of statistics in one sentence, the following attempt by Barnett (1973) is praiseworthy: "*We define statistics as the study of how information should be employed to reflect on, and give guidance for action in, a practical situation involving uncertainty.*" This definition contains several crucial elements that have to be taken into account when reflecting upon the nature of statistics. "Information" emphasises that in statistics we work with data. "Uncertainty" provides an indication of the reality which usually confronts us in practice: our conclusions from a statistical study can rarely be made with complete confidence. The phrase "... give guidance for action in ..." reflects the truth that statistics is often applied in decision making. Whatever definition of statistics we use, one claim is universally acknowledged: statistics is essentially an interdisciplinary field and has had a profound impact on virtually all branches of science, medicine, industry and government.

One of the earliest developments in the field of statistics was the method of least squares, developed in the early 1800's by Legendre. This was followed by developments in the field of probability theory, and by the early twentieth century major advances were being made in the fields of multivariate analysis and experimental design. Since approximately the beginning of the 1970's, however, probably the major impetus for growth in the field of statistics has been advances in computer technology.

Calculating has always been an integral part of the task of a statistician. Throughout history, statisticians have consequently been keen and intensive users of the most modern available calculating aids. It is quite amusing to read accounts of the experiences of statisticians with some early versions of the modern computer. F.N. David refers in a 1989 interview to the year 1933. At that time she was research assistant to the famous Karl Pearson, and an important part of her work entailed calculations for tables of correlation coefficients. For this purpose she made use of the most modern available computing technology, a Brunsviga computer. She says: "*I estimated that I turned that hand Brunsviga roughly two million times. We used to use Brunsvigas and they would carry tens in one register, but they wouldn't carry tens in another. Before I learned how to manipulate long knitting needles, which was strictly illegal, I was always jamming the damned thing. When you jammed it, you were supposed to go tell the professor and then he would tell you what he thought of you; it was really rather awful.*" (Laird, 1989). One of the first electronic computers was the so-called Colossus, built around 1943 in England to assist in deciphering secret German codes. I.J. Good recounts that the Colossus was typically capable of up to 100 billion binary calculations before something significant would go wrong

(Banks, 1996). W.J. Dixon was for many years involved in development of the BMD statistics software (later to become BMDP). In a 1993 interview he recalls the IBM 7090 mainframe computer taken into use in 1970 at UCLA. At that time this was purported to be one of the most powerful computers in the world. Dixon declares however: "*Today's PC (or laptop) has memory capacity 50 times, memory access 300 times and disk storage 5000 times the 7090*" (Flournoy, 1993). If we keep in mind that Dixon expressed this opinion in 1993, it is clear that today we find personal computers in many modern homes that are much more powerful than the best computers of 30 years ago.

The computer technology available today was almost unimaginable just two short decades ago. Today, computers can with relative ease analyse large quantities of complex data using statistical programs – even on personal computers. The computer has completely changed the meaning of the term statistical analysis, and consequently has caused a revolution in the field of data analysis. Many statistical theories for data analysis were developed long before the arrival of computers, but these methods were not widely known outside the field of theoretical statistics – simply because the computational power to perform complex calculations was not available. Today, however, any interested researcher has access to all the resources needed to address almost any kind of research problem or data analysis. In fact, many of the most successful current methodologies (a prime example being the bootstrap) would have been impractical without the increased computing power that has become available.

The advent of computers has not only had an impact on data analysis, but also on the ease with which data can be dealt with: in fact, computers have completely changed

the way in which data can be collected and stored. The large scale data sets and sheer volumes of data that statisticians are faced with today, pose a new set of research challenges. Although the statistician of today has a much larger arsenal of tools available for solving a data analysis problem, the range of problems that has to be confronted has also widened dramatically. A quote in this regard from the Preface of Hastie et al. (2001) seems appropriate: "*We are drowning in information and starving for knowledge.*"

The topic investigated in this thesis should be viewed from the perspective sketched above. The *support vector machine* is a technique that was developed approximately 10 years ago mainly in the machine learning community. It is a numerically intensive technique that can be used in the analysis of huge data sets. Although it has hitherto received limited attention from statisticians, this situation is changing. The intention in this thesis is therefore to provide a brief introduction to the topic of support vector machines, and to report on its application in a practical situation.

2. Data mining: new problems and new techniques

It was mentioned in the previous section that one of the most important changes in the field of computer science (and also in the field of statistics) over the past two decades has been in the quantities of data becoming available and being stored. This has largely been responsible for the origin of the relatively new field known as data mining.

Data mining is an inter-disciplinary area in statistics and machine learning. It comprises a collection of techniques that can be used to find meaningful patterns in large quantities of data. Berry and Linoff (2000) define data mining as "... the process of exploration and analysis, by automatic or semi-automatic means, of large quantities of data in order to discover meaningful patterns and rules". As with statistics, advances in the field of data mining were also made possible by rapid growth in computing power. Not so long ago, most data mining techniques would not have been feasible; today, most companies have access to large data sets and employ data miners in some or other capacity.

The Electronic Statistics Textbook quotes Pregibon[1] and says that data mining is "a blend of statistics, AI [artificial intelligence] and database research". Without a doubt, data mining has grown from several fields of study, two of the most important probably being machine learning and statistics. The goal of machine learners can be described as getting computers to learn by example, and consequently displaying some form of intelligence. Researchers in machine learning have contributed many important algorithms to aid in recognising patterns that exist in data, but their understanding of the theoretical basis of these techniques has been greatly aided by statisticians.

Data mining uses many traditional statistical techniques (for example regression) for exploratory data analysis. However, it also encompasses techniques that are not considered as traditional by the statistical community. A goal of a technique such as neural networks is not necessarily to understand the structure and relationships in a

[1] Pregibon, D. (1997). Data Mining. Statistical Computing and Graphics, 7, 8.

data set, but rather to make predictions based on the data set. The embracing of such techniques by data miners has also raised the profile of these techniques in the statistics community.

Another example of a technique that has gained from both the machine learning and the statistics communities is the support vector machine. This method is very popular in the machine learning and computer science communities, and has recently started to benefit from the input of statisticians, who are contributing to an understanding of the properties of the method.

Today, data mining is a rapidly growing area, and is also recognised by the statistics community as a major area of research.

3. Overview of the thesis

In this chapter we have emphasised how the advent of computers has lead to certain advances in the field of statistics, and has given rise to new concepts in statistics being explored through cooperation with scientists in the field of machine learning. It also examined the concept of data mining.

Chapter 2 will examine some of the more traditional tools for classification and regression that are currently available in most commercial data mining and / or statistics software packages. Discriminant analysis, the linear model, classification and regression trees and neural networks will be discussed.

Chapter 3 is an in-depth look at one of the newer tools available to statisticians and data miners, namely the support vector machine (SVM). The basic concept of an SVM will be explained for the simple binary classification case where we have linearly separable data; extensions to the basic SVM algorithm will then be explored for the case where the data are not linearly separable (the soft margin approach and the kernel trick will be introduced to deal with this), multi-class classification, and also an extension to an SVM algorithm for regression. Lastly, certain advantages and disadvantages of support vector machines will be discussed and we will mention some fields where support vector machines have been successfully applied.

The computing environment for implementing the more traditional techniques as well as support vector machines is discussed in Chapter 4. A brief overview of some of the larger commercial software packages for data mining is given, and then a particular freeware product – the *R* package – is discussed in some detail.

Empirical results of two specific research problems in a life assurance environment will be presented in Chapters 5 and 6. Chapter 5 consists of results of an empirical study for a classification problem, namely to predict whether an insurance client was likely to lapse any policy. Techniques that are used to model the problem are linear discriminant analysis, classification trees and support vector machines. We will see that SVMs performed better than the more traditional techniques in predicting lapses.

Results of an empirical study for a regression problem can be found in Chapter 6. Here the problem is to estimate the household income of insurance clients.

Techniques that are used are ordinary multiple regression, regression trees and support vector machines and we will once again see that SVMs generally performed better.

Some concluding remarks will be made in Chapter 7. It will also touch on some areas that warrant further work and briefly look at some other new methods that are available for classification and regression in a data mining context.

Finally in this chapter, a few remarks on the notation that will be used in the thesis. Vectors will be denoted by using boldface, for example $\mathbf{x}$. All vectors will be column vectors, unless specifically indicated otherwise. The transpose of a vector (or matrix) will be denoted in the usual way, for example $\mathbf{x}'$. Matrices will typically be denoted by capital boldface letters. Other specific notational conventions will be explained as required.

Chapter 2

Traditional methods for classification and regression

1. Introduction to classification

In a statistical classification problem we observe a response variable (also called the output or dependent variable), Y, together with predictor variables (also called input or independent variables) $X_1, X_2, \ldots, X_p$. The response variable is qualitative or categorical, and we will denote its categories by $\{1, 2, \ldots, K\}$. We assume that the predictor variables have been observed for N sample cases, and the resulting data set will be denoted by $\mathcal{T} = \{(\mathbf{x}_i, y_i), i = 1, 2, \ldots, N\}$. Here $\mathbf{x}_i$ is a $p-$component vector representing the values of $X_1, X_2, \ldots, X_p$ for the i-th sample case. Our purpose is to use the data $\mathcal{T}$, also called the *training data*, to determine a rule that can be used to assign a new case with observed values of the predictor variables in the vector $\mathbf{x}$ to one of the available categories or classes. Obviously we would like this assignment to be as accurate as possible. In this regard we refer to the *training error*

of a classification procedure when we have in mind the proportion of incorrect classifications if the procedure is applied to all the cases in $\mathcal{T}$. Although a small training error is desirable, and procedures are frequently developed in such a way that the training error is minimised, it is the so-called *generalisation error* or *test error* of a classification procedure that determines its real worth. The generalisation error of a classification procedure is the probability (under some assumed probability distribution generating the data) that a new case, i.e. a case not forming part of the training data, will be incorrectly classified. It is important to realise that small training error does not necessarily correspond to small generalisation error. In this regard the phenomenon of *overfitting* should be kept in mind. A procedure overfits a given training data set if it follows the peculiar characteristics of this data set too closely. This typically results in small training error, but large generalisation error – an undesirable state of affairs. Overfitting is a real danger, especially in high-dimensional cases (i.e. cases where p is large), and various techniques have been devised to guard against it.

Many statistical techniques have been developed to deal with practical classification problems. Well-known examples are different forms of discriminant analysis, logistic regression, neural networks, and classification trees. In the following sections linear discriminant analysis, classification trees and neural networks will be reviewed briefly as procedures that can be used for statistical classification. These are the procedures that will be applied in Chapter 5 to a data set from an insurance environment in order to compare the generalisation errors of these procedures with that of a support vector machine.

2. Classification: linear discriminant analysis

2.1 Background, and Fisher's linear discriminant function

Discriminant analysis was introduced by the famous English statistician R.A. Fisher in 1936. There are different versions of discriminant analysis, such as linear discriminant analysis, quadratic discriminant analysis, flexible discriminant analysis, regularised discriminant analysis, and canonical discriminant analysis. In this chapter the focus will be on linear discriminant analysis. The interested reader is referred to Hastie et al. (2001, Chapters 4 and 12) for a discussion of the various alternative forms of discriminant analysis. Since our application of linear discriminant analysis in Chapter 5 will be for a case where there are only two groups into which an entity can be classified, we will restrict our discussion here to the two-group case.

Consider, therefore, a binary classification problem with classes Π_1 and Π_2. We assume that the training data $\mathcal{T}$ consists of two parts: a random sample $\mathcal{T}_1 = \{\mathbf{x}_{11},...,\mathbf{x}_{1N_1}\}$ of size N_1 from Π_1, and a random sample $\mathcal{T}_2 = \{\mathbf{x}_{21},...,\mathbf{x}_{2N_2}\}$ of size N_2 from Π_2. The sample mean vectors are given by $\bar{\mathbf{x}}_i = \frac{1}{N_i}\sum_{j=1}^{N_i}\mathbf{x}_{ij}$, $i = 1,2$.

Fisher's intuitive approach to the problem of finding a suitable classification function from the training data was to look for the linear function $\mathbf{v}'\mathbf{X}$ of the predictor variables which best separates the observations from the two groups when these observations are thus linearly transformed. This led him to look for the vector $\tilde{\mathbf{v}}$ which maximises the ratio

$$J(\mathbf{v}) = \frac{\mathbf{v}'\mathbf{B}\mathbf{v}}{\mathbf{v}'\mathbf{W}\mathbf{v}} = \frac{\langle \mathbf{B}\mathbf{v}, \mathbf{v} \rangle}{\langle \mathbf{W}\mathbf{v}, \mathbf{v} \rangle} \tag{2.1}$$

where the between class scatter matrix $\mathbf{B} = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'$, and the within class scatter matrix $\mathbf{W} = \sum_{i=1}^{2} \sum_{j=1}^{n_i} (\mathbf{x}_{ij} - \bar{\mathbf{x}}_i)(\mathbf{x}_{ij} - \bar{\mathbf{x}}_i)'$. Maximising the ratio $J(\mathbf{v})$ in (2.1) leads to the direction which maximises the separation between the two classes, whilst minimising the variance within each class in this direction. It can be shown that the vector which maximises $J(\mathbf{v})$ is the eigenvector of $\mathbf{W}^{-1}\mathbf{B}$ corresponding to the largest eigenvalue of this matrix, and that this eigenvector is given by $\tilde{\mathbf{v}} = c\mathbf{W}^{-1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)$, with the constant $c \neq 0$ arbitrary. Without loss of generality we choose $c = 1$. A new case with predictor variable vector $\mathbf{x}_0$ can now be classified by calculating $\langle \tilde{\mathbf{v}}, \mathbf{x}_0 \rangle = \tilde{\mathbf{v}}'\mathbf{x}_0 = (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)' \mathbf{W}^{-1}\mathbf{x}_0$, and classifying the case into class Π_1 if $\langle \tilde{\mathbf{v}}, \mathbf{x}_0 \rangle$ is closer to $\langle \tilde{\mathbf{v}}, \bar{\mathbf{x}}_1 \rangle$ than to $\langle \tilde{\mathbf{v}}, \bar{\mathbf{x}}_2 \rangle$, and into class Π_2 otherwise. This is easily seen to be equivalent to classifying into class Π_1 if $\left\langle \tilde{\mathbf{v}}, \mathbf{x}_0 - \frac{1}{2}(\bar{\mathbf{x}}_1 + \bar{\mathbf{x}}_2) \right\rangle > 0$, and into class Π_2 otherwise. The function $\tilde{\mathbf{v}}'\mathbf{X}$ is called *Fisher's linear discriminant function.*

2.2 A parametric approach

In the approach described above no assumption was made about the probability distribution of the predictor vector $\mathbf{X}$. It is interesting that the same discriminant rule can be derived for the two-group case by following a parametric approach and applying Bayes' rule. We now briefly present this approach.

Let $f_i(\mathbf{x})$ be the probability density function of $\mathbf{X}$ in Π_i, $i=1, 2$, and suppose we have a prior probability π_i associated with class Π_i, $i=1, 2$, i.e. before observing any data we feel that an entity has probability π_i to belong to class Π_i, $i=1, 2$. Application of Bayes' rule then yields the following so-called Bayes classification procedure: classify an entity with observed predictor vector $\mathbf{x}$ into Π_1 if

$$\frac{\pi_1 f_1(\mathbf{x})}{\pi_1 f_1(\mathbf{x})+\pi_2 f_2(\mathbf{x})} > \frac{\pi_2 f_2(\mathbf{x})}{\pi_1 f_1(\mathbf{x})+\pi_2 f_2(\mathbf{x})} \tag{2.2}$$

and into Π_2 otherwise. Suppose we now assume that the underlying population densities are multivariate normal with respective mean vectors $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ and common covariance matrix $\boldsymbol{\Sigma}$, i.e.

$$f_i(\mathbf{x}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)'\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)\right\} \tag{2.3}$$

If we further assume equal prior probabilities, then after simplification the classification rule (2.2) becomes: classify an entity with observed predictor vector $\mathbf{x}$ into Π_1 if and only if $\delta_M(\mathbf{x},\boldsymbol{\mu}_1) < \delta_M(\mathbf{x},\boldsymbol{\mu}_2)$, where $\delta_M(\mathbf{x},\boldsymbol{\mu}_i) = (\mathbf{x}-\boldsymbol{\mu}_i)'\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)$ is the population Mahalanobis distance of $\mathbf{x}$ from the mean vector $\boldsymbol{\mu}_i$, $i=1, 2$. We therefore classify into the population with minimum Mahalanobis distance to the observed predictor vector $\mathbf{x}$. It is now easy to show that this is equivalent to classifying $\mathbf{x}$ into Π_1 iff

$(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)'\boldsymbol{\Sigma}^{-1}\left\{\mathbf{x} - \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)\right\} > 0$. In practice the population parameters appearing in these expressions are of course unknown and have to be estimated from the available sample data. If we replace these parameters by their usual unbiased estimates, the last expression above becomes $(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'\mathbf{W}^{-1}\left\{\mathbf{x} - \frac{1}{2}(\bar{\mathbf{x}}_1 + \bar{\mathbf{x}}_2)\right\} > 0$, which is identical to the classification rule based on Fisher's linear discriminant function described at the end of Section 2.1.

2.3 Some other forms of discriminant analysis

Linear discriminant analysis as described above has been extended in several ways in the literature. These extensions include quadratic discriminant analysis, flexible discriminant analysis, regularised discriminant analysis, and canonical discriminant analysis. Hastie et al. (2001) provide details on these extensions. In this section we only briefly refer to some of the ways in which the linear discriminant model can be extended.

In the above approach, it was assumed that the covariance matrices of the predictor random vector in the different groups were equal. When this is not the case, quadratic discriminant analysis rather than the linear version should be used. The reader is referred to Hastie et al. (2001, pp. 88-90) for more detail on this topic. In other cases we have prior information suggesting that an observation is more likely to come from one of the groups than the others. Such prior information can be incorporated into our analysis and in principle we can make use of the Bayes classifier in (2.2). We specify the prior probability π_i that an observation comes from Π_i, and $\mathbf{x}$ is then classified

as coming from group Π_i if $\pi_i f_i(\mathbf{x}) = \max_j \pi_j f_j(\mathbf{x})$. If the prior probabilities are unknown quantities, their values are typically estimated by the ratios N_i/N. Finally, when the costs associated with different wrong classifications are not all the same, a decision theoretic approach can be used.

2.4 Optimality of linear discriminant analysis

For linear discriminant analysis as described above, we make the rather restrictive assumption that all the observations come from multivariate normal distributions and that the covariance matrices of these distributions are equal. When these assumptions hold, linear discriminant analysis is in some sense "optimal". Fatti et al. (1982) state that the total probability of misclassification is minimised by the linear discriminant function if sampling is from multivariate populations with known means and known equal covariances matrices. However, if sampling is from multivariate normal populations with known means and known but unequal covariance matrices, the total probability of misclassification will be minimised by the quadratic discriminant function. When the means and common covariance matrix are not known, these quantities can be estimated from the data and as long as the sample size is reasonable linear discriminant analysis should then still give satisfactory results.

3. Introduction to regression

Regression analysis is probably the most widely used statistical technique. Regression models are used to study the relationship between a quantitative response variable and a set of input or independent variables. For a training sample $\mathcal{T} = \{(\mathbf{x}_i, y_i), i = 1, 2, \ldots, N\}$ consisting of observations of the predictor variables together with corresponding observations of the response, a regression model usually has the form $y_i = f(\mathbf{x}_i) + \varepsilon_i$, $i = 1, 2, \ldots, N$. In this expression the ε_i's are typically assumed to be uncorrelated error (noise) terms with zero mean and common variance σ^2. The purpose in regression analysis is to estimate the unknown function f from the training data. Note that $E(\varepsilon) = 0$ implies that $E(Y \mid \mathbf{X} = \mathbf{x}) = f(\mathbf{x})$, so that we are actually trying to estimate the conditional expectation of the response random variable Y.

If the form of the function f is known (except for the value of certain unknown parameters), we have a parametric regression model. If we also know that f is linear in its unknown parameters, the model is referred to as *a linear model*. An additional assumption that is frequently made for a linear model is that the error term $\boldsymbol{\varepsilon}$ is normally distributed.

The regression surface expresses the best prediction of the response variable Y in terms of the independent variables $\mathbf{x}_i$. Naturally, it is usually not possible to perfectly determine the relationship, and the deviation of a particular point from its predicted value is called the residual value. These residual values can be used to devise a criterion for determining the best fitting surface. In least squares estimation

problems, the best fitting regression surface is calculated to minimise the sum of the squared deviations of the observed points from their predicted values.

The multiple linear regression model can be conveniently expressed in matrix notation. We write $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, where $\mathbf{y}$ is the $N-$component vector containing the observations that were made of the response, $\mathbf{X}$ is an $N \text{x} (p+1)$ matrix with first column containing 1's (this provides for an intercept term in the linear model), and with remaining columns containing the observed values of the predictor variables, $\boldsymbol{\beta}$ is a $(p+1)-$component vector of unknown parameters which have to be estimated, and $\boldsymbol{\varepsilon}$ is an $N-$component vector containing the error terms. The method of least squares for estimating $\boldsymbol{\beta}$ can now be described in matrix terms. For a given value of $\boldsymbol{\beta}$ the vector of residuals is given by $\mathbf{r}(\boldsymbol{\beta}) = \mathbf{y} - \mathbf{X}\boldsymbol{\beta}$, with corresponding sum of squares $\mathbf{r}(\boldsymbol{\beta})'\mathbf{r}(\boldsymbol{\beta})$. The least squares estimate of $\boldsymbol{\beta}$ is the vector $\mathbf{b}$ minimising this sum of squares. It easily follows from simple calculus that $\mathbf{b}$ satisfies the so-called normal equations given by $(\mathbf{X}'\mathbf{X})\mathbf{b} = \mathbf{X}'\mathbf{y}$ and, provided that the matrix $(\mathbf{X}'\mathbf{X})$ is non-singular, we can obtain $\mathbf{b}$ from $\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$. If all the necessary assumptions for a linear model are met, the least squares estimates $\mathbf{b}$ are the best linear unbiased estimates of the unknown parameters (in the sense that they have minimum variance in the class of estimators that are unbiased and are linear functions of the responses). This result is known as the Gauss-Markov theorem.

The least squares estimates $\mathbf{b}$ can now be used to estimate the conditional expected response at given values of the predictor variables, or to predict the response at such values of the predictor variables. For the $\mathbf{x}$-vectors in the training data set we have

the vector of predicted response values $\hat{\mathbf{y}} = \mathbf{Xb} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$. Similarly, for a new given set of predictor variable values $\mathbf{x}$, the least squares prediction of the response is given by $\hat{y} = \mathbf{b}'\mathbf{x}$.

The residual or error sum of squares is given by $SSE = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$, and it is easy to show that $SSE = \mathbf{y}'\mathbf{y} - \mathbf{y}'\mathbf{Xb}$. The variance, σ^2, of the error term in the model is estimated unbiasedly by the mean residual sum of squares, i.e. $\hat{\sigma}^2 = \frac{SSE}{N - p - 1}$. If we now assume that the error random variables, and therefore also the response random variables, are normally distributed, confidence intervals can be constructed for the individual parameters in the model. Hypothesis tests concerning these quantities can also be performed.

Of course in the above discussion we assumed that $(\mathbf{X}'\mathbf{X})^{-1}$ exists; that is, that the matrix $\mathbf{X}$ is of full column rank. In practice, this is sometimes not the case. The solution for this scenario is derived in much the same way as described above, with the exception that instead of using $(\mathbf{X}'\mathbf{X})^{-1}$ (which does not exist for a model not of full rank), we use instead any so-called generalised inverse of $(\mathbf{X}'\mathbf{X})$. For details, see Searle (1971).

We close this section by referring briefly to some extensions of the multiple linear regression model described above. Firstly, this model can be transformed into the general linear model (also called the multivariate linear model). In doing so, provision can be made for additional response variables. Allowance is also made for linear transformations of linear combinations of multiple response variables. That is,

the vector $\mathbf{y}$ of N observations of a single response variable can be replaced by a matrix $\mathbf{Y}$ containing N observations of m different response variables. Such a general linear model is given in matrix terms by $\mathbf{YM} = \mathbf{XB} + \mathbf{E}$ where $\mathbf{M}$ is an $m \times s$ matrix of coefficients defining s linear transformations of the response variables.

Of course, a linear model as described above is based on some very strong assumptions. In particular, it is assumed that the relationship between the response variable and the input variables is linear (or nearly linear). The least squares estimates of the regression coefficients are very unstable in the presence of multicollinearity of the sample data. Some solutions that have been suggested when these assumptions are violated, are ridge regression and robust regression. Ridge regression is a modification of the least squares method for regression to allow biased estimators of the regression coefficients. It is used to stabilise the estimated regression coefficients in the presence of multicollinearity; for details, see Marquardt and Snee (1975).

Robust regression procedures are available for cases where the errors follow a non-normal distribution; these procedures are designed to dampen the effect of outliers that would otherwise have a strong influence on the analysis. A number of different robust regression procedures exist; they can be classified into three general classes, namely M estimators, R estimators and L estimators. For details, see for example Mendenhall and Sincich (1996).

When the relationship between the response variable and the input variables does not appear to be linear, methods such as polynomial regression or other nonlinear

regression techniques can be used. Unequal variances among the error terms can be remedied by employing a weighted least squares procedure.

4. Classification and regression trees

4.1 Basic idea and terminology

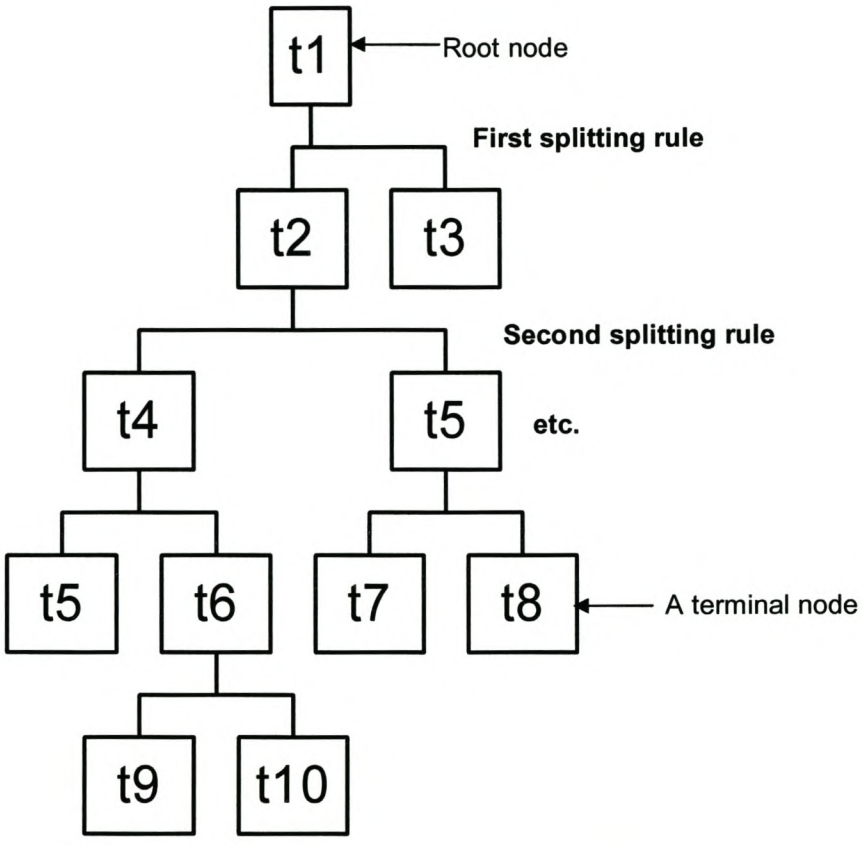
The basic analytical idea of decision trees originally springs from work in the social sciences by Morgan and Sonquist (1963), and Morgan and Messenger (1973). Breiman et al. (1984) brought trees to the attention of the statistical community and proposed new algorithms for constructing trees. Trees can be used for classification and regression purposes; here, the concept of trees will be explained in the classification context.

Consider once again the training data set $\mathcal{T} = \{(\mathbf{x}_i, y_i), i = 1, 2, \ldots, N\}$, with data in two or more classes, with a qualitative response variable and a number of input or predictor variables. The goal of a classification tree is to segment the data, through recursive partitioning, into subgroups that are as homogeneous or pure as possible with respect to the response variable. At each point where the data are split into additional subgroups, it is said that a split (or decision) is made.

Structurally, a decision tree consists of a number of nodes. The root is the top node of the tree. The root node is split into purer nodes, and in turn these nodes are then split up into further nodes, and the process continues in this way until no more splits are possible, and the maximal tree has been constructed or grown, or until some stopping criterion is satisfied. A node at which no further splits are made, is called a terminal node, or leaf, and at such a node no further splits are attempted. A non-terminal node,

on the other hand, will contain a question or rule on which a split can be based. Each leaf will contain the label of a classification.

Graphically a hierarchical classification tree could look as follows:



Additional concepts that need to be introduced at this stage are those of parent nodes and child nodes. This can easily be done by considering the above graphical presentation of a decision tree: nodes t5 and t6 are considered to be child nodes of parent node t4; similarly, node t6 is the parent node of child nodes t9 and t10.

4.2 Elements in the construction of a tree

There are some basic elements around which the entire construction of a classification tree revolves:

- which splits should be considered
- deciding which split is best
- when should splitting be stopped (i.e. when should a node be considered terminal)
- assigning a class to each terminal node

We comment briefly on each of these aspects.

4.2.1 Splits that should be considered

Usually there are a large number of splits that could be considered at each node of the tree, and it would be very time-consuming to consider all possible splits. For example, if a predictor variable x is a categorical variable with K categories, then there are $2^{K-1} - 1$ possible different splits for this variable. Therefore, to prevent the number of splits that should be considered from becoming enormous, the possible splits that are considered are usually restricted. A very common restriction is to consider only binary splits. This is generally preferred to multiway splits, since multiway splits fragment the data too quickly, leaving insufficient data at the next level of the tree. Furthermore, multiway splits can be reproduced by a series of binary splits.

4.2.2 Deciding which split is best

For all the possible splits that will be considered, a measure is needed of how to decide when a split is best.

In some cases the decision will be simple: if the response is the same in all child nodes as in the parent node, the split is essentially worthless. On the other extreme, a split can result in pure child nodes, and this means that the split is undoubtedly best.

However, usually things are not as clear-cut, and therefore splits are evaluated by considering the diversity or, more accurately, the reduction in diversity achieved by each possible split. The reason for this is that we want each subgroup to be as pure or homogeneous as possible. Therefore, we need a clearly defined way of measuring the diversity in each group.

There are several ways of calculating the measure of diversity. One very popular method is called the Gini index. The Gini index is basically a measure of the probability that any two elements of the population chosen through random sampling with replacement will belong to two different classes. If we have K classes, the formula for the Gini index is given by $\sum_{k=1}^{K} \hat{p}_{mk}\left(1-\hat{p}_{mk}\right)$ where $\hat{p}_{mk}$ is the proportion of class k observations in node m.

Other popular diversity measures are cross-entropy, which is given by $\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$,

the Pearson chi-squared test statistic, the twoing and ordered twoing criterion, and the symmetric Gini index (which is an adaptation of the Gini index in order to incorporate misclassification costs).

Once the diversity measure has been calculated, a high diversity value indicates that the distribution of the classes in the set is even, while a low value means that a single class dominates.

Different splitting criteria will generally give different results for a data set under consideration, and there is no single best choice. The Gini index, for example, favours splits that group training data cases from the majority class in one branch of the tree, while the entropy criterion favours balanced splits. In most software packages for decision trees, the user can choose which splitting criterion to use, and this should be done by carefully considering which gives the best results for the data set under consideration.

4.2.3 When to stop splitting nodes

The simplest case of tree construction is when there exists an exact partition of the data set, or in other words, when every case in the data set can be correctly classified. In this case, the tree is simply grown until every example is correctly classified. In practice, this is often not the case, and such a problem is called a noisy classification problem. There are two alternatives in this instance: stop growing the tree early, or prune the tree after constructing an overly large tree.

The complexity of a decision tree depends on its number of leaves. A tree can be continually split until all leaves are pure – such a tree would fit the training data perfectly, but would not generalise very well (i.e. it would not do a good job of classifying new cases). One of the reasons for this is that, as the tree is grown, the sample size at each of the nodes where the splits are made becomes smaller and smaller, and meaningless patterns might be detected. At the other extreme, a tree could have only a root node, and no other leaves. Every case would then have the same predicted value. Generally, neither of the two extremes will be a very good classification tree.

There are two approaches to deciding the appropriate size of the tree, namely pre-pruning and post-pruning. Pre-pruning involves the use of stopping rules to stunt the growth of a tree, and is often also called bonsai techniques. Post-pruning, on the other hand, involves cutting back certain branches of the tree after the tree has been grown.

A generally used pre-pruning rule is to stop growing at a node if the node is pure. However, this generally does not result in sufficient stunting of growth. Therefore, various tests are often applied at each node to determine whether any further splits will be useful. These tests may be quite simple, such as requiring a minimum number of cases in each node, or it can be more complicated, for instance subjecting the proposed splits to appropriate significance tests.

A problem with pre-pruning is that it relies only on the training data and is actually unreliable as a means of preventing overfitting. However, requiring a fairly large minimum node size can still result in a well-behaved tree, since this usually means

that the sample at each node is more representative of the population. Another potential problem with pre-pruning is that it runs the risk of missing future splits that occur below weak splits. An advantage of pre-pruning is that it is computationally less demanding.

Post-pruning (often just called "pruning") involves growing a large tree and pruning back branches. A pruned tree is basically a subset of the fully-grown decision tree. By pruning, branches of the tree that are deemed unnecessary for the tree to generalise well are pruned off the tree. An algorithm that has been proposed is to find the classification error rate with each smaller and smaller sub-tree of the initial tree. To avoid error rates that keep dropping as the tree becomes more complex, a complexity term is added to the error, so that greater complexity is punished. In this way a branch is only kept when the improvement in classification performance outweighs the drawback of introducing greater complexity to the model. However, this technique, known as cost-complexity pruning, does not necessarily work well with large data sets. The best way to apply pruning in practice on large data sets is to measure the performance of the tree and all its sub-trees on a test data set. The tree can then be pruned back to the one that minimises the error on the test set. This approach can be improved even further by using several test sets, and the optimal decision tree can be chosen as the one that performs the most consistently over all the test sets.

4.2.4 Assigning a class to each terminal node

Class assignment is an easy and logical process: the class that is the most represented in a node is the class that will be assigned to that node. For example, if class 1 has the highest proportion of cases in node t, class 1 will be assigned to node t.

4.3 Tree algorithms and software

The most popular decision tree algorithms are CART (Classification And Regression Trees) and CHAID (CHi-squared Automatic Interaction Detection).

The CART algorithm was proposed by Breiman et al. (1984). It is restricted to binary splits and uses post-pruning by v-fold crossvalidation (see Section 4.1 of Chapter 3 for a complete definition of v-fold crossvalidation). All possible binary splits are considered. For very large data sets, within-node sampling can be used. Criteria for multiclass problems and regression trees are also available.

CHAID is a modification of the AID algorithm originally developed by Morgan and Sonquist (1963) and Kass (1980). CHAID uses multiway splits and pre-pruning for growing classification trees. The best multiway split is found through a stepwise agglomerative algorithm. Splitting and stopping criteria are based on the statistical significance of the chi-squared test.

4.4 Some advantages of trees

- Trees can handle missing values naturally and easily.
- Trees are easy to interpret, and yield easily interpretable rules. This is one of the main reasons for the popularity of trees, especially in medical and biological applications.
- Trees can detect interactions between variables (i.e. variables that contain encoded knowledge of the dependent variable).
- Trees can automatically select input variables (i.e. they can prioritise independent variables).
- Trees are insensitive to a difference in scale between inputs.
- Classification and regression using trees are non-parametric techniques that make no assumptions about the underlying distribution of the data. Consequently, trees are quite robust.
- Trees can use any type of data (i.e. categorical and numerical variables).
- Trees can handle large data sets with many variables.
- Trees perform classification and also provide estimates of misclassification probability.
- Trees can easily be adapted for multi-class classification and regression problems.

5. Neural networks

5.1 The foundations of neural networks

Neural networks started receiving considerable attention in the statistical community during the early 1990's. They can be used for regression and classification, and have been applied successfully in areas where regression models and other related statistical techniques are traditionally used.

Neural networks are named as such because these networks attempt to model the capabilities of the human brain. The human brain consists of millions of neurons that are interconnected. A neuron receives a signal from other neurons; if the signal received is determined to be greater than a certain threshold level, the neuron is activated and "fires" (i.e., it sends a signal to other neurons). Of course, the real power of the human brain lies in its ability to learn, and also its ability to generalise. The human brain can also recognise patterns in the presence of noise and withstand localised damage. Neural networks try to mimic these properties.

In an artificial neural network, the parallel to the neurons of the human brain are units, or nodes. The nodes are also connected, and by applying an activation function to the inputs, outputs are generated. The initial aim in the development of neural networks was to combine many simple computing elements into a highly interconnected system, analogous to the human brain, in order to mimic complex phenomena such as intelligence. More recently however, statistical methods and numerical analysis have been incorporated into neural networks.

5.2 A basic neural network

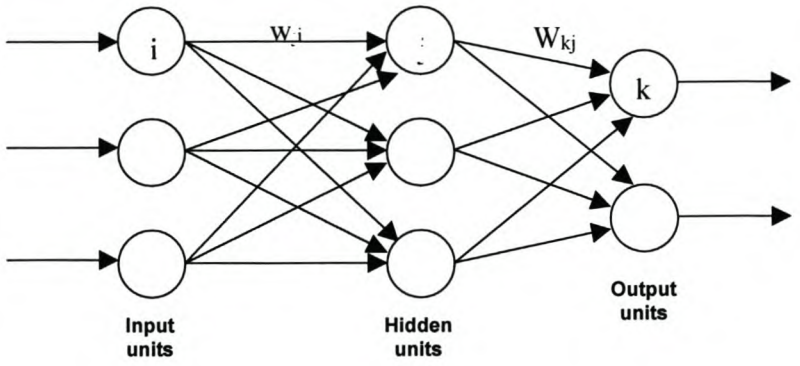
5.2.1 Network architecture

Feed-forward neural networks form a class of flexible, nonlinear regression, discriminant and data reduction models. They are useful if no mathematical formula is known that relates the inputs to the outputs, if prediction is more important than explanation, and if there is a large amount of training data. A neural network consists of input units, hidden units and output units. Input units obtain the values of the input variables and can standardise them if necessary. The hidden units perform the internal calculations and provide the non-linearity that makes neural networks useful. The output units compute the predicted values and compare them to the values of the dependent variables. Each input and hidden unit produces a computed value which is passed forward. The values computed by the output units are the predicted values, which are used to calculate the error which training attempts to minimise. Connections between units have weights associated with them, and the training process tries to minimise the error by iteratively adjusting the weights. Most units also have one or two numeric values associated with them, called the bias and the altitude. These are also estimated parameters that are adjusted during the training process.

The "standard" form of a neural network is a fully connected feed-forward network with one hidden layer. It has been shown (Cybenko, 1989) that such a network with a sufficient number of hidden units can approximate any continuous function to any

degree of accuracy. (In a feed-forward neural network, connections between units are unidirectional only, i.e. loops are not allowed.)

Such a "standard" neural network can be illustrated diagrammatically as follows:



In other words, each of the input units is connected to each of the hidden units, and each of these connections has an associated weight. For instance, input unit i is connected to hidden unit j through a weight w_{ji}, where w_{ji} represents the strength of the connection between input unit i and hidden unit j. Similarly, each of the hidden units is connected to the output units; for example, hidden unit j is connected to output unit k through weight W_{kj}. Because this is a feed-forward network, units in the same layer are not connected to each other, and connections are unidirectional only.

What happens in a network such as the one illustrated above is that the input values are passed to the input units in the input layer. Usually the inputs are scaled to be in the range [-1, 1]. The input units perform no operation on the input values presented to them, but simply pass these on to the hidden units. The net input into a unit in the hidden layer is therefore a combination of the outputs from the input units. The

hidden unit then applies an activation function to its net input, and this is then passed through to the output units. The output units therefore receive a net input from the hidden units, which is again a combination of the outputs from the hidden units, and in turn outputs a quantity. The inputs are combined into a net input by way of a combination function. There are two kinds of combination functions that are commonly used, namely linear combination functions and radial combination functions. Linear combination functions compute a linear combination of the weights and the values feeding into the unit and add a constant (intercept) value. Radial combination functions compute the squared Euclidean distance between the vector of weights and the vector of values feeding into the unit and multiply this by a squared intercept value.

The values produced by the combination function are transformed by an activation function. The activation function involves no weights or other estimated parameters. Some of the most commonly used activation functions are the identity function (which does not change the value of the argument, and whose range is potentially unbounded), sigmoid functions (S-shaped functions such as logistic and hyperbolic tangent functions that produce bounded values within the range 0 to 1 or –1 to 1), the Softmax function (which is a generalisation of the logistic function that affects several units together, forcing the sum of their values to be 1), and exponential and reciprocal functions (which are bounded below by 0 but unbounded above).

Networks are trained by minimising an error function. Some of the more commonly used error functions are those based on the normal distribution, Huber M-estimators, redescending M-estimators, the gamma distribution, the Poisson distribution, the

Bernoulli distribution, entropy, multiple Bernoulli, and multiple entropy. The choice of error function will depend on the type of dependent variable.

The only remaining problem before training can start is to assign the weights w_{ji} that connect the input units to the hidden units, and W_{kj} that connect the hidden units to the output units. This is where the learning property of neural networks comes into play.

Initial values of the weights are usually assigned randomly. A wide variety of training techniques can then be used. Some conventional techniques are standard batch backpropagation, standard incremental backpropagation, Quickprop, RPROP, Levenberg-Marquardt, Quasi-Newton and conjugate gradient techniques.

When a neural network is presented with training data, it compares the output values obtained from the network to the known (desired) answers. The error function is then used to adjust the weights in the network, and this process is repeated until the weights no longer change significantly and the error therefore no longer decreases.

The above learning process is called supervised learning, because the target output values for the training data are known. The output from the network is compared to the expected output, and feedback is given to the network to correct possible errors. Unsupervised learning is also possible for neural networks. In these instances, the target output values will be unknown, and the network discovers correlations and similarities among the data of its own accord.

Two standard forms of neural networks are multilayer perceptrons (MLP) and radial basis function (RBF) networks.

Multilayer perceptrons are networks that can have any number of inputs and one or more hidden layers with any number of units. Linear combination functions are used in the hidden and output layers and a sigmoidal activation function is used in the hidden layers. Any number of outputs is possible with any activation function. It has connections between the input layer and the first hidden layer, between hidden layers and between the last hidden layer and the output layer. Given enough data, enough hidden units and enough training time, a MLP with just 1 hidden layer is a universal approximator (which means that it can theoretically approximate any continuous surface to any degree of accuracy).

RBF networks also have any number of inputs, but typically only one hidden layer with any number of units. They use radial combination functions in the hidden layer, based on the squared Euclidean distance between the input vector and the weight vector. It typically uses an exponential or Softmax activation function in the hidden layer, and uses linear combination functions in the output layer, which can have any number of outputs with any activation function. It has connections between the input layer and the hidden layer, and between the hidden layer and the output layer.

A final note needs to be made about the generalisation ability of a neural network. It is often the case that neural networks have too many weights and will therefore overfit the data. One of the earliest methods suggested to prevent overfitting is the use of early stopping rules. This means that the model is trained only for a while, and that

training is stopped before a global minimum error is reached. Another method that has been suggested is weight decay, which is similar to ridge regression in the case of linear models. A penalty is added to the error function, which has the effect of shrinking the weights toward zero. For more details, see Hastie et. al (2001).

5.2.2 Mathematical representation of a simple neural network

To illustrate mathematically the working of a neural network, the simple case of a two-layer feed-forward neural network with backpropagation will be discussed. A sigmoid activation function will be used, together with the identity combination function and the sum of squares error function. For more details, the reader is referred to Warner and Misra (1996).

Suppose we have a neural network with N input units, M hidden units, and O output units. Each of the N input units is connected to each of the M hidden units, with weights w_{ji} (where i is the ith input unit and j is the jth hidden unit). In turn, each of the M hidden units is connected to each of the O output units, with weights W_{kj} (j is the jth hidden unit and k is the kth output unit).

Let x_{pi} be the value of the ith input for the pth observation. Then the net input into hidden unit j is given by $h_{pj} = \sum_{i=1}^{N} w_{ji} x_{pi}$. Hidden unit j applies an activation function to its net input, and outputs $v_{pj} = g(h_{pj}) = \dfrac{1}{1+e^{-h_{pj}}}$.

Output unit k receives a net input $f_{pk} = \sum_{j=1}^{M} W_{kj} v_{pj}$, and outputs the quantity $\hat{y}_{pk} = g(f_{pk}) = \frac{1}{1+e^{-f_{pk}}}$. Now the weights need to be found such that the sum of squared errors is minimised.

Using the sum of squared errors as error function means that the error is given by

$$E = \frac{1}{2}\sum_{p=1}^{n}\sum_{k=1}^{O}(y_{pk} - \hat{y}_{pk})^2,$$

where the index p refers to the pth observation, and there is a total of n observations. Also, y denotes the observed response, and $\hat{y}$ the response predicted by the model.

We now have to minimise the error E. This quantity is a function of the weights w_{ji} and W_{kj}. Therefore, the partial derivative of E with respect to a weight represents the rate of change of E with respect to that weight (i.e., the slope of E). Moving the weights in a direction down the slope will result in a decrease in E (i.e., in a decrease in the error). Thus we need to calculate

$$\Delta W_{kj} = -\eta \frac{\partial E}{\partial W_{kj}}$$

and

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}},$$

which can be shown to equal

$$\Delta W_{kj} = -\eta[(-1)(y_{pk} - \hat{y}_{pk})]\hat{y}_{pk}(1 - \hat{y}_{pk})v_{pj}$$

and

$$\Delta w_{ji} = \eta \sum_{k=1}^{O} (y_{pk} - \hat{y}_{pk})\hat{y}_{pk}(1 - \hat{y}_{pk})W_{kj}v_{pj}(1 - v_{pj})x_{pi}$$

respectively. The weights from the hidden units to the output units are updated by

$$W_{kj}^{t+1} = W_{kj}^{t} + \Delta W_{kj}$$

(i.e., if the current estimate of the weight is W_{kj}^{t}, we add the weight adjustment ΔW_{kj} to obtain an updated weight estimate W_{kj}^{t+1}), and similarly for the w_{ji} weights. These adjustments are then repeated until the optimal weights are obtained.

Chapter 3

Support vector machines

1. Background

"Support vector machines are a very specific class of algorithms, characterised by the use of kernels, the absence of local minima, the sparseness of the solution and the capacity control obtained by acting on the margin, or on other 'dimension independent' quantities such as the number of support vectors." (Cristianini and Shawe-Taylor, 2001).

The basic theory behind the concept of support vector machines was developed in Russia during the sixties, with Vapnik and co-workers' 'Generalised Portrait' algorithm (Vapnik and Lerner, 1963, Vapnik and Chervonenkis, 1964).

The present form of the support vector machine, however, was developed at AT&T Bell Laboratories by Vapnik and co-workers, and was formalised by Boser et al. (1992), and introduced at the Computational Learning Theory (COLT) 1992 conference. In this chapter we provide a brief introduction to the topic of support

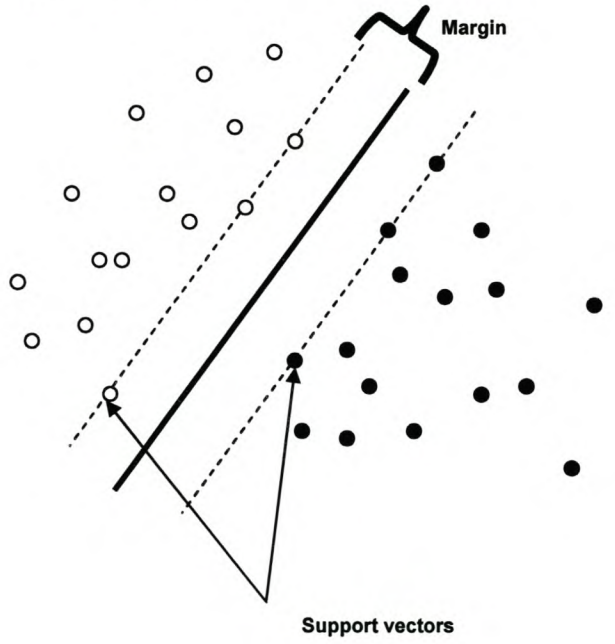
vector machines. We will see that support vector machines can be used for classification as well as regression problems, and that a support vector machine often depends only on a subset of the training data. It is in this sense that we refer to the *sparseness* of a support vector algorithm. Much more complete and detailed discussions of support vector machines can be found in Schölkopf and Smola (2002), and Kroon (2003).

2. The basic principle underlying a support vector machine

The basic concept of a support vector machine (SVM) can best be explained by looking at the simplest form thereof, namely the SVM for binary classification, where the training data are linearly separable.

Consider therefore once again the two-class classification problem introduced at the start of Chapter 2. We have a training data set $\mathcal{T} = \{(\mathbf{x}_i, y_i), i = 1, 2, \ldots, N\}$, consisting of observations on a categorical response, Y, together with predictor variables $X_1, X_2, \ldots, X_p$. In this chapter we initially assume that $Y \in \{-1, 1\}$. We assume that N_1 cases from population 1 have been included in the training data, and N_2 cases from population 2, where $N_1 + N_2 = N$. In addition, we assume that the training data from the two classes are linearly separable, i.e. there exists a hyperplane $\{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle + b = 0\}$ in $p-$dimensional input space that perfectly separates the training data cases from the two populations. This implies that we can find a vector $\mathbf{w} \in R^p$ and a scalar b such that $\langle \mathbf{x}_i, \mathbf{w} \rangle + b < 0$ for all the cases from population 1,

and $\langle \mathbf{x}_i, \mathbf{w} \rangle + b > 0$ for all the cases from population 2. In fact, under these circumstances we have infinitely many such separating hyperplanes. The support vector algorithm looks for the so-called *optimal separating hyperplane* between the two classes. To define what this means, we first have to define the concept of the *margin* of a data set with respect to a given hyperplane. From results presented in Appendix A, we know that the signed distance from a point $\mathbf{z} \in R^p$ to a given hyperplane $\{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle + b = 0\}$ is given by $\dfrac{\langle \mathbf{z}, \mathbf{w} \rangle + b}{\|\mathbf{w}\|}$. Now suppose we find the minimum absolute distance that a training data point lies from the hyperplane. This quantity is called the margin of the training data set with respect to the given hyperplane. Expressed mathematically we therefore have $\text{margin} = \min\left\{ y_i \left(\dfrac{\langle \mathbf{x}_i, \mathbf{w} \rangle + b}{\|\mathbf{w}\|} \right) : \mathbf{x}_i \in \mathrm{T} \right\}$ (keeping in mind that $y_i \in \{-1, 1\}$). This is illustrated diagrammatically below:



The optimal separating hyperplane is now defined to be the hyperplane which maximises the margin. The points lying on the boundaries are called the support vectors.

There are several arguments in favour of finding a hyperplane that maximises the margin – see for example Schölkopf and Smola (2002, pp.192-196). It seems that there is good reason to believe that maximising the margin leads to a classifier with good generalisation properties, i.e. a classifier that has small generalisation error. We briefly refer to one of these arguments. Generalisation error refers to the properties of the trained support vector machine when it is applied to new data cases. Suppose such new data cases are generated within hyper-spheres centred at the training data points. Then maximising the margin is equivalent to maximising the radius of such hyper-spheres while preserving perfect separation of the two populations.

3. Mathematical representation of support vector machines

Consider our training data set $\mathcal{T} = \{(\mathbf{x}_i, y_i), i = 1, 2, \ldots, N\}$, consisting of observations on a categorical response, Y, together with predictor variables $X_1, X_2, \ldots, X_p$. As indicated previously we assume that $y_i \in \{-1, 1\}$. If the training data are linearly separable, it means that we can find a hyperplane such that all the $\mathbf{x}_i$'s for which $y_i = 1$ lie on one side of the hyperplane, and all the $\mathbf{x}_i$'s for which $y_i = -1$ lie on the opposite side of the hyperplane. Furthermore, because any hyperplane H in $p-$dimensional input space can be written as

$$H = \{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle + b = 0\}$$

(where $\mathbf{w}$ is a vector orthogonal to the hyperplane), and because the training data are linearly separable, it means that we can find a vector $\tilde{\mathbf{w}}$ and a scalar $\tilde{b}$ such that

$$y_i(\langle \tilde{\mathbf{w}}, \mathbf{x}_i \rangle + \tilde{b}) \geq \varepsilon_+ \; \forall \, i \in \{j : y_j = 1\}$$

and

$$y_i(\langle \tilde{\mathbf{w}}, \mathbf{x}_i \rangle + \tilde{b}) \geq \varepsilon_- \; \forall \, i \in \{j : y_j = -1\}.$$

If we let $\varepsilon = \min(\varepsilon_-, \varepsilon_+)$, then this implies that $y_i(\langle \tilde{\mathbf{w}}, \mathbf{x}_i \rangle + \tilde{b}) \geq \varepsilon \;\; \forall \, i$. Hence, if we let $\mathbf{w} = \frac{1}{\varepsilon}\tilde{\mathbf{w}}$ and $b = \frac{1}{\varepsilon}\tilde{b}$, we find that

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \tag{3.1}$$

for $i \in \{1, \ldots, N\}$, with equality holding for at least one $\mathbf{x}_i$.

From (3.1) it is now clear that the margin of the training data set with respect to the hyperplane $H = \{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle + b = 0\}$ constructed above is given by

$$\min\left\{ y_i \left(\frac{\langle \mathbf{x}_i, \mathbf{w} \rangle + b}{\|\mathbf{w}\|} \right) : \mathbf{x}_i \in \mathcal{T} \right\} = \frac{1}{\|\mathbf{w}\|}.$$

The support vector algorithm now looks for the separating hyperplane with the largest margin. In other words, we need to maximise $1/\|\mathbf{w}\|$. This can be done by minimising $\|\mathbf{w}\|$, or, for convenience sake, minimising $\frac{1}{2}\|\mathbf{w}\|^2$.

To minimise $\frac{1}{2}\| \mathbf{w} \|^2$, subject to the constraints (3.1), we introduce Lagrange multipliers $\alpha_1,...,\alpha_N$, to obtain the Lagrangian

$$L(\mathbf{w},b,\boldsymbol{\alpha}) = \frac{1}{2} < \mathbf{w},\mathbf{w} > - \sum_{i=1}^{N} \alpha_i \{ y_i (< \mathbf{w},\mathbf{x}_i > + b) - 1 \}. \tag{3.2}$$

The Lagrangian function now needs to be minimised with respect to $\mathbf{w}$ and b, while simultaneously maximised with respect to $\boldsymbol{\alpha}$, subject to the constraints that $\alpha_i \geq 0$. To do this, we need to take partial derivatives of L, and then set these derivatives equal to zero:

$$\frac{\partial L(\mathbf{w},b,\boldsymbol{\alpha})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i = 0 \tag{3.3}$$

$$\frac{\partial L(\mathbf{w},b,\boldsymbol{\alpha})}{\partial b} = \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{3.4}$$

Solving (3.3) for $\mathbf{w}$ yields

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i x_i \ , \tag{3.5}$$

thereby expressing the optimal $\mathbf{w}$ in terms of the training data and the still to be determined optimal values of the Lagrange multipliers $\alpha_1,...,\alpha_N$. From the results in Appendix A, we know that the primal optimisation problem above can be transformed to a dual problem, which is often easier to solve. We proceed as follows.

Substituting (3.4) and (3.5) into the Lagrangian L, gives the Wolfe dual problem: maximise with respect to $\alpha_1,...,\alpha_N$

$$L_D(\boldsymbol{\alpha}) = \frac{1}{2} < \{\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i\}, \{\sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j\} > - \sum_{i=1}^{N} \alpha_i \{y_i (< \{\sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j\}, \mathbf{x}_i > + b) - 1\}$$

$$= \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j < \mathbf{x}_i, \mathbf{x}_j > - \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j < \mathbf{x}_i, \mathbf{x}_j > - b \sum_{i=1}^{N} \alpha_i y_i + \sum_{i=1}^{N} \alpha_i$$

$$= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j < \mathbf{x}_i, \mathbf{x}_j >, \tag{3.6}$$

subject to the constraints $\alpha_i \geq 0$ and (3.4). The expression for the dual Lagrangian is now in quadratic form, while the constraints are all linear in the α_i's. This is therefore a quadratic programming problem which can be solved quite easily using standard optimisation techniques, thereby obtaining the optimal values of $\alpha_1,...,\alpha_N$.

To calculate the optimal value of b, the Karush-Kuhn-Tucker (KKT) conditions must be introduced (see Appendix A). The KKT conditions simply state that at the optimal solution, the product between the dual variables and the constraints have to vanish. The relevant Karush-Kuhn-Tucker conditions for the primal Lagrangian L are

$$\alpha_i \left[y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 \right] = 0 \qquad \forall\ i, \tag{3.7}$$

Therefore $\alpha_i = 0$ unless $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 = 0$. This means that only those data points which lie closest to the hyperplane can have non-zero α_i's. All these points for which $\alpha_i > 0$ are called the *support vectors*. It is an interesting property of support vector machines that the final procedure in no way depends on the training data points which are not support vectors, i.e. if we retrain a support vector machine on the training data without the data points which are not support vectors, we obtain exactly the same solution as before. Also interesting is the relationship between the number of support vectors and the expected test error as given by Schölkopf and Smola (2002) (see their Proposition 7.4 on page 198).

Let $\boldsymbol{\alpha}^*$ now be the vector of optimal Lagrange multipliers, subject to the given constraints, and suppose $\mathbf{w}^*$ and b^* are the parameters of the corresponding optimal hyperplane. Then from (3.5) we have $\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* y_i x_i$. The KKT-conditions in (3.7) also give $y_i(\langle \mathbf{w}^*, \mathbf{x}_i \rangle + b^*) = 1$. Therefore, because $y_i = \pm 1$, we see that the optimal value b^* can be calculated from

$$b^* = y_i - \langle \mathbf{w}^*, \mathbf{x}_i \rangle. \tag{3.8}$$

For reasons of numerical stability, b^* is usually calculated as the mean of the values actually calculated from (3.8) for all positive α_i's.

The optimal separating hyperplane has now been determined. Once this has been done, new data points can be classified by determining on which side of the

separating hyperplane they lie. In other words, the class of a new case with predictor vector $\mathbf{x}$ is taken to be $sign(\langle \mathbf{w}^*, \mathbf{x} \rangle + b^*)$, or

$$sign(\sum_{i=1}^{N} \alpha_i^* y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b^*). \tag{3.9}$$

This is also referred to as a "hard" classifier for new cases. Gunn (1998) suggests the use of a "soft" classifier, which instead gives a real valued output between -1 and 1 when a point is within the margin, where there are no training data. In other words, instead of using the decision function in (3.9) above, we can use,

$f(\mathbf{x}) = h(\langle \mathbf{w}^*, \mathbf{x} \rangle + b^*)$,

where $h(z) = \begin{cases} -1 & for\ z < -1 \\ z & for\ -1 \le z \le 1 \\ +1 & for\ z > 1. \end{cases}$

4. Extensions of the basic support vector algorithm

4.1 Overlapping data – the soft margin approach

In practice data are seldom linearly separable, and a single deviant point can acutely affect the support vector machine hyperplane. The first option to handle non-linearly separable data is to introduce an additional cost associated with the misclassification of data cases into the objective function that has to be optimised. This is a suitable option when it is expected, or possibly even known, that a hyperplane can correctly separate the data.

The additional cost associated with misclassification is introduced by relaxing the constraints in (3.1) through the introduction of positive slack variables ξ_i, $i = 1, 2, \ldots, N$ (Cortes and Vapnik, 1995). This allows the constraints in (3.1) to be violated, subject to the proviso that any violation will lead to an increase in the objective function.

This approach leads to the constraints in (3.1) becoming

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i$$

for $i = 1, 2, \ldots, N$, and with $\xi_i \geq 0 \quad \forall i$.

If the ξ_i's are made large enough, the constraints above will always be met; it is therefore important to ensure that large values of ξ_i are in turn penalised in the objective function. For an error to occur the relevant ξ_i must be greater than 1, which means that $\sum_i \xi_i$ is an upper bound on the number of training errors.

These considerations lead to the following objective function that has to be minimised:

$$\frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle + CF(\sum_{i=1}^{N} \xi_i),$$

where C is a cost parameter chosen by the user, with a larger C corresponding to assigning a higher penalty to errors, and F is a monotonically increasing function, usually chosen as $F(x) = x$ to ensure that the minimisation of the objective function remains a quadratic programming problem.

To solve the resulting quadratic programming problem, additional Lagrange multipliers η_i are introduced for the constraints $\xi_i \geq 0$. The primal Lagrangian becomes

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\eta}) = \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \{ y_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 + \xi_i \} - \sum_{i=1}^{N} \eta_i \xi_i .$$

Taking partial derivatives with respect to $\mathbf{w}$, b and ξ_i gives the expressions which should be set to zero:

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\eta})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i = 0$$

$$\frac{\partial L(\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\eta})}{\partial b} = \sum_{i=1}^{N}\alpha_i y_i = 0$$

and

$$\frac{\partial L(\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\eta}))}{\partial \xi_i} = C - \alpha_i - \eta_i = 0\,.$$

The solution to $\mathbf{w}$ is therefore still given by $\mathbf{w} = \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i$, while the constraint $\sum_{i=1}^{N}\alpha_i y_i = 0$ is augmented by the additional constraint $C = \alpha_i + \eta_i$, $i = 1,2,\ldots,N$.

The Wolfe dual objective function now becomes

$$L_D(\boldsymbol{\alpha}) = \frac{1}{2}\left\langle \{\sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i\}, \{\sum_{j=1}^{N}\alpha_j y_j \mathbf{x}_j\}\right\rangle + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\eta_i\xi_i - \sum_{i=1}^{N}\alpha_i\{y_i(\left\langle \sum_{j=1}^{N}\alpha_j y_j \mathbf{x}_j\,,\,\mathbf{x}_i\right\rangle + b) - 1 + \xi_i\}$$

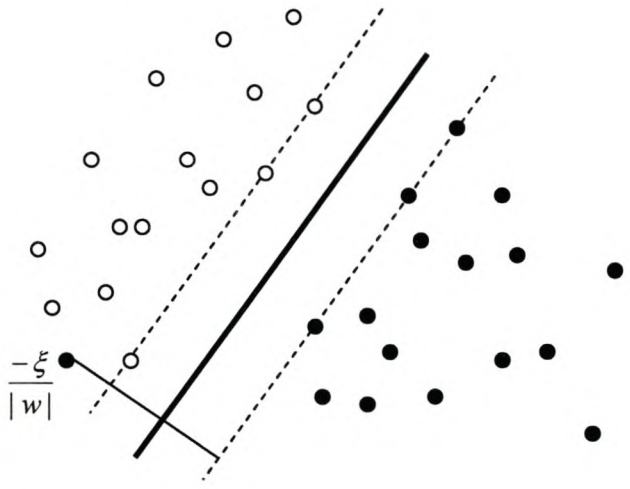
$$= \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j\left\langle \mathbf{x}_i,\mathbf{x}_j\right\rangle + \sum_{i=1}^{N}C\xi_i - \sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j\left\langle \mathbf{x}_i,\mathbf{x}_j\right\rangle - b\sum_{i=1}^{N}\alpha_i y_i + \sum_{i=1}^{N}\alpha_i - \sum_{i=1}^{N}\alpha_i\xi_i - \sum_{i=1}^{N}\eta_i\xi_i$$

$$= \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j\left\langle \mathbf{x}_i,\mathbf{x}_j\right\rangle + \sum_{i=1}^{N}(C-\alpha_i-\eta_i)\xi_i$$

$$= \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j\left\langle \mathbf{x}_i,\mathbf{x}_j\right\rangle,$$

which is identical to the dual objective function for the case of linearly separable data. In other words, changing the primal objective function has no effect on the dual objective function. The only difference is that in this case there is the additional constraint $0 \leq \alpha_i \leq C$, i.e. the only difference from the linearly separable case is that the α_i now have an additional upper constraint C.

Graphically, the problem looks as follows:



Now an optimal $\boldsymbol{\alpha}$ can be found, and then the corresponding $\mathbf{w}^*$ and b^*, in exactly the same way as previously.

The uncertain part of this approach is that the cost parameter C must be specified (ultimately to reflect the knowledge of the noise in the data). So far, no clear-cut solution for determining C has been suggested. The parameter is usually found by means of cross-validation. In v-fold cross-validation the training data set is divided into v equally sized subsets and the first subset is tested using the classifier trained on

the remaining $v-1$ subsets. This process is repeated for all v subsets, and for a range of values of C. The value of C which minimises the total cross-validation test error is then finally used in the optimisation on the full training data set.

Meyer (2002) suggests starting with extreme values such as $C=1$ and $C=1000$, and then using cross-validation to further determine the optimum value for C. Gunn (1998) states that the useful range of C lies between the point where all the Lagrange multipliers are equal to C and where only one of them is bounded by C, while Schölkopf and Smola (2002) use $C/N=10$ as a default specification, where N is the number of training data cases.

As C decreases, the width of the margin increases. If C tends to infinity, the SVM solution will tend towards the solution obtained by the optimal separating hyperplane. On the other hand, if C tends to zero, the SVM solution will converge to the solution where the emphasis is simply on maximising the margin, and less on minimising the misclassification error. In other words, a large width margin will be created.

Gunn (1998) uses the well-known iris data set and finds that the values of $C=1$ and $C=100$ seem to offer good solutions, depending on the type of decision boundary that is desired. He concludes that the parameter C may have more than one optimal value, and that some degree of prior knowledge about the problem should be incorporated into the decision on the final value of C.

Another variation on the soft margin approach for SVMs is the more natural ν-parametrisation (Schölkopf and Smola, 2002). In this approach the constant C is

replaced by a parameter $\nu \in (0,1]$ which provides an upper and lower bound for the fraction of margin errors (i.e. points with non-negative slack variables) and the fraction of examples that will be chosen as support vectors respectively. The quadratic optimisation problem to be solved for $\nu-$SV classification is derived by Schölkopf and Smola and is given by :

$$\text{maximize } -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle$$

subject to the constraints

$$0 \le \alpha_i \le \frac{1}{N}, \quad \sum_{i=1}^{N}\alpha_i y_i = 0 \text{ and } \sum_{i=1}^{N}\alpha_i \ge \nu .$$

Compared to the ordinary $C-$classification dual problem, we find that there is an extra constraint, which introduces additional difficulty especially in the case of large data sets. Some approaches have been suggested to overcome this problem (Chang and Lin, 2001; Pérez-Cruz et al., 2001). Chang and Lin (2001) also give a complete account of the connection between $\nu-$SVMs and $C-$SVMs.

4.2 Kernel trick – the non-linearly separable case

The above formulation of the SVM algorithm works for data that can be well separated by a linear decision boundary in input space. When a linear boundary is inappropriate, it might be necessary to non-linearly transform the data in input space to feature space to ensure linear separability of the data in that space. An optimal separating hyperplane is therefore constructed in feature space.

Motivation for such an approach is provided by Cover's theorem, which states that in many cases where data are not linearly separable in a low dimensional (input) space, it may become linearly separable if it is transformed to a higher dimensional (feature) space. The theorem basically characterises the number of possible linear separations of N points in $p-$dimensional space. If $N \leq p+1$ then 2^N separations are possible; however, if $N > p+1$ the number of possible linear separations is equal to $2\sum_{i=0}^{p}\binom{N-1}{i}$. In other words, larger p (or more dimensions) means that more separations are possible. Therefore transforming the input data to feature space increases the probability of obtaining a linearly separable problem.

To do this, so-called reproducing kernels are used; this means that instead of using dot or inner products (as in the case of the linear SVM), we substitute a kernel function. This will yield a linear decision surface in feature space, which corresponds to a non-linear decision surface in input space. The mathematical motivation for using kernels is Mercer's Theorem, which states that for certain mappings φ from input to feature space and any two points $\mathbf{u}$ and $\mathbf{v}$ in input space, the inner product of the mapped points can be evaluated using a kernel function without ever explicitly knowing the mapping, i.e. $\langle \varphi(\mathbf{u}), \varphi(\mathbf{v}) \rangle = K(\mathbf{u}, \mathbf{v})$ for some kernel function $K(.,.)$.

Therefore, instead of maximising (3.6) , we now maximise

$$\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{3.10}$$

subject to the constraints $\alpha_i \geq 0$, $i = 1,2,\ldots,N$, and (3.4). Note that the inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ in (3.6) has been replaced by the kernel function expression $K(\mathbf{x}_i, \mathbf{x}_j)$ in (3.10).

The optimal coefficient vector, $\mathbf{w}^*$, can be expressed as before in terms of the training data and the optimal values of the Lagrange multipliers, viz.

$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* y_i \varphi(\mathbf{x}_i). \tag{3.11}$$

This expression is typically not very useful, however, since it contains the feature mapping, φ, which maps the input space into a feature space, and it may in fact be that the latter is infinite dimensional. However, we are only interested in the decision function $sign(\langle \mathbf{w}^*, \varphi(\mathbf{x}) \rangle + b^*)$, and if we substitute the expression in (3.11) into the decision function, we obtain

$$sign(\left\langle \sum_{i=1}^{N} \alpha_i^* y_i \varphi(\mathbf{x}_i), \varphi(\mathbf{x}) \right\rangle + b^*) = sign(\sum_{i=1}^{N} \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}) + b^*), \tag{3.12}$$

where b^* is once again obtained as the mean over all support vectors of

$$\begin{aligned} b^* &= y_i - \langle \mathbf{w}^*, \varphi(\mathbf{x}_i) \rangle \\ &= y_i - \left\langle \sum_{j=1}^{N} \alpha_j^* y_j \varphi(\mathbf{x}_j), \varphi(\mathbf{x}_i) \right\rangle \end{aligned}$$

$$= y_i - \sum_{j=1}^{N} \alpha_j^* y_j \left\langle \varphi(\mathbf{x}_j), \varphi(\mathbf{x}_i) \right\rangle$$

$$= y_i - \sum_{j=1}^{N} \alpha_j^* y_j K(\mathbf{x}_j, \mathbf{x}_i).$$

We therefore see that the decision function in (3.12) can be calculated from knowledge of the kernel function, $K(.,.)$, alone, without the feature mapping φ needing to be known. This very important result is known as the *kernel trick*, and it makes possible extension of the support vector algorithm to procedures that may be highly non-linear in input space.

The choice of kernel function is crucial to ensure that the resulting support vector machine will generalise well. It will always be possible to map the input space into a dimension greater than the number of training points and to produce a classifier with no classification errors on the training set. However, this will not generalise well. The correct choice of kernel is therefore very important. The choice of kernel also basically amounts to model selection, and prior knowledge of a problem can be used to help with choosing a suitable kernel. Model selection is then basically reduced to adjusting the kernel parameters. Presently, methods such as bootstrapping and cross-validation are the preferred methods for kernel parameter selection.

Following Gunn (1998), we now summarise important kernel functions that may be used. Note that these functions are all positive definite – a characterising property of kernel functions.

i) **Polynomial**

Suppose d is an integer value, usually 2 or 3. Then there are two options:

- $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle^d$
- $K(\mathbf{x}_1, \mathbf{x}_2) = (\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1)^d$

Usually the second kernel is preferred.

ii) **Gaussian Radial Basis Function**

$$K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\sigma^2})$$

iii) **Exponential Radial Basis Function**

$$K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|}{2\sigma^2})$$

This kernel produces a piecewise linear solution and is suitable when discontinuities are acceptable.

iv) **Multi-layer Perceptron**

$$K(\mathbf{x}_1, \mathbf{x}_2) = \tanh(\rho \langle \mathbf{x}_1, \mathbf{x}_2 \rangle + \vartheta))$$

This is the kernel representation of a multi-layer perceptron with one hidden layer (i.e. a neural network). The support vectors of a solution obtained by using this kernel correspond to the first layer, and the Lagrange multipliers to the weights.

v) **Fourier series**

$$K(\mathbf{x}_1, \mathbf{x}_2) = \frac{\sin\left((N + \frac{1}{2})(\mathbf{x}_1 - \mathbf{x}_2)\right)}{\sin(\frac{1}{2}(\mathbf{x}_1 - \mathbf{x}_2))}$$

vi) **Splines**

$$K(\mathbf{x}_1, \mathbf{x}_2) = \sum_{r=0}^{\kappa} \mathbf{x}_1^r \, \mathbf{x}_2^r + \sum_{s=1}^{N} (\mathbf{x}_1 - \tau_{\mathbf{x}})^{\kappa} + (\mathbf{x}_2 - \tau_{\mathbf{x}})_+^{\kappa}$$

vii) **B splines**

$$K(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{B}_{\mathbf{2N+1}}(\mathbf{x}_1 - \mathbf{x}_2)$$

viii) **Additive kernels**

$$K(\mathbf{x}_1, \mathbf{x}_2) = \sum_i K_i(\mathbf{x}_1, \mathbf{x}_2)$$

This form of a kernel function is justified because the sum of two positive definite functions is also positive definite.

ix) **Tensor product**

$$K(\mathbf{x}_1, \mathbf{x}_2) = \prod_i K_i(\mathbf{x}_1, \mathbf{x}_2)$$

Here multidimensional kernels are formed by tensor products of kernels.

The most popular and frequently used kernel functions are the polynomial, radial basis and multi-layer perceptron (sigmoid) kernel functions. Stecking and Schebesch

(2002) state that Radial Basis Function (RBF) kernels seem least "pathological" due to their flexible localisation properties and the fact that $K(\mathbf{x},\mathbf{x})=1$. Hsu et al. (2003) also recommend the RBF kernel as a reasonable first choice, since it can handle non-linearity, needs fewer hyperparameters than the polynomial kernel and also causes less numerical difficulties. Furthermore, the linear kernel is a special case of the RBF kernel, while the sigmoid kernel also behaves like the RBF kernel for certain parameter values. A disadvantage of sigmoid kernels is that the sigmoid kernel is not valid for some parameter specifications.

5. Multi-class classification for SVMs

The SVM architecture considered up to now has been limited to the case of binary classification. However, in many real-world problems, there can be more than two classes into which an object can be classified. Several approaches have been suggested for adapting support vector machines to multi-class problems. The two main approaches are:

i) Combine several binary class problems to form a multi-class problem. Here the most often used techniques are one-against-all, one-against-one and Directed Acyclic Graph (DAG) SVMs.

ii) Consider all classes of data at once. This means a change in the underlying structure of the SVM. Techniques used are, amongst others, k-class SVMs and k-class linear programming machines.

5.1 Combinations of binary classification SVMs

5.1.1 One-against-all

Suppose we have data in k classes. The one-against-all approach requires construction of k SVM models. The ith SVM will be trained with all examples in the ith class having positive labels, and all other examples having negative labels. In more detail, consider the training data set $\mathcal{T} = \{(\mathbf{x}_i, y_i), i = 1, 2, \ldots, N\}$, consisting of observations on the categorical response, Y, together with predictor variables. In this section we assume that $Y \in \{1, 2, \ldots, k\}$. The training data consist of N_i observations from population i, where $N_1 + N_2 + \ldots + N_k = N$. We find the i^{th} SVM by solving a quadratic programming optimisation problem exactly as before, but with all the training data cases from population i acting as one group, and all the other cases acting as the second group. Proceeding in this way we obtain k classification functions of the form $\langle \mathbf{w}_i^*, \varphi(\mathbf{x}) \rangle + b_i^*$, $i = 1, 2, \ldots, k$. Normally in a two-group scenario our procedure would now entail taking the sign of such a classification function. However, to classify a new case with input vector $\mathbf{x}$ into one of the k available classes, we now calculate these classification functions and $\mathbf{x}$ is then classified into the class which has the largest value of the classification function, i.e.

$$class\ of\ x \quad = \quad \arg\max_{i=1,\ldots,k} \left[\langle \mathbf{w}_i^*, \varphi(\mathbf{x}) \rangle + b_i^* \right].$$

The one-against-all technique has several advantages, but also some disadvantages. The technique is easy to implement and displays fairly good performance empirically. However, depending on the difficulty of each individual one-against-all problem, the

output range of the different SVMs will differ. Ways around this include normalising the SVM outputs and feeding the SVM outputs into a single layer perceptron. This method also does not take into account the mutual exclusivity of class membership. This results in a great loss in discriminative power when there is no class i for which the probability that $\mathbf{x}$ is an element of class i is greater than 0.5.

5.1.2 One-against-one (pairwise classification)

The basic idea here is as follows. If we have data in k classes, we construct $k(k-1)/2$ classifiers, each of which will train data from two classes only. By solving the appropriate quadratic programming optimisation problems, we obtain $k(k-1)/2$ classification functions of the form $\left\langle \mathbf{w}_{ij}^{*}, \varphi(\mathbf{x}) \right\rangle + b_{ij}^{*}$, $i, j = 1, 2, \ldots, k; i \neq j$. To classify a new case with predictor vector $\mathbf{x}$, the following voting strategy is used: if $sign(\left\langle \mathbf{w}_{ij}^{*}, \varphi(\mathbf{x}) \right\rangle + b_{ij}^{*})$ indicates that $\mathbf{x}$ is an element of class i rather than class j, then the total vote for class i is increased by one; otherwise the vote for the j^{th} class is increased by one. In the end, $\mathbf{x}$ is predicted to be in the class with the largest number of votes. This voting approach is called a 'Max Wins' strategy.

The one-against-one technique also has its advantages and disadvantages. The number of SVMs that need to be trained grow quadratically with the number of classes. However, the optimisation problems encountered are much smaller, since only data from the two classes involved are used for training each SVM. A drawback, however, is that smaller samples mean higher variance, so generalisation performance of the individual SVMs may not be good.

5.1.3 Directed Acyclic Graphs (DAGSVM)

The training phase for this procedure is the same as for the one-against-one method, where we solve $k(k-1)/2$ binary SVMs. However, in the testing phase, this method uses a rooted binary directed acyclic graph with $k(k-1)/2$ internal nodes and k leaves. Each node is a binary SVM for the i^{th} and j^{th} classes. Given a test data vector $\mathbf{x}$, starting at the root node, a binary decision function is evaluated. It then moves left or right, depending on the output value. Therefore, we go through a path before reaching a leaf node which indicates the predicted class. For this procedure some analysis of generalisation ability can be done, and testing time is less than for the one-against-one method (see Hsu and Lin, 2001).

5.1.4 Other techniques

Some other techniques that are used to solve the multi-class problem by means of appropriate combination of the output from binary SVMs are based on error-correcting output coding (see Schölkopf and Smola, 2002), and game theory.

5.2 All data classes at once

Weston and Watkins (1998) suggest solving $k-$class problems directly by modifying the SVM objective function to allow for the computation of a multi-class classifier. The optimisation problem is generalised to the following:

$$\text{minimise } \phi(\mathbf{w},\xi) = \frac{1}{2}\sum_{h=1}^{k}\langle \mathbf{w}_h, \mathbf{w}_h \rangle + C\sum_{i=1}^{m}\sum_{h\neq y_i}\xi_i^h$$

subject to the constraints

$$\langle \mathbf{w}_{y_i}, \phi(\mathbf{x}_i)\rangle + b_{y_i} \geq \langle \mathbf{w}_m, \phi(\mathbf{x}_i)\rangle + b_m + 2 - \xi_i^m$$

$$\xi_i^m \geq 0,\ i = 1,\ldots,l, m \in \{1,\ldots,k\} \setminus y_i .$$

The accuracy of results obtained by this approach is comparable to those obtained by the one-against-all approach. However, it has the disadvantage that all support vectors have to be dealt with simultaneously, while with the other methods smaller sets of support vectors are considered at a time. This decreases training time.

Hsu and Lin (2001) compare $k-$class support vector machines as well as another method proposed by Crammer and Singer (2000) with one-against-all, one-against-one and DAGSVM. They find that for large problems the one-against-one and DAG methods are more suitable for practical use than the other methods, although the all-together methods in general need fewer support vectors. Schölkopf and Smola (2002) also state that, although the choice of approach needs to take into consideration factors such as the required accuracy, the time available for development and training

and the nature of the classification problem, the one-against-all approach often produces acceptable results.

In general, though, the choice of method for multi-class classification would depend on the required accuracy, the time available for training as well as the number of classes.

Other methods for multi-class classification not mentioned here include an SVM based on a uniform convergence result (Guermeur et al., 2000), and a $k-$class linear programming machine (Weston and Watkins, 1998).

6. Support vector regression

6.1 Basic idea

The extension of the concept of support vector machines to the case of regression was introduced by Vapnik et al. (1997). SVMs are extended to regression problems by using an alternative loss function, which is modified to include a measure of distance. The hyperplane fitted in feature space is seen as the estimator of the regression function, and through the loss function we penalise deviations of data points from this hyperplane (this is somewhat similar to the cost function introduced with the soft margin approach).

6.2 Loss functions

The loss function usually used for support vector regression is Vapnik's ε-insensitive loss function, which is given by

$$l_\varepsilon = \begin{cases} 0 & \text{for } |f(\mathbf{x}) - y| < \varepsilon \\ |f(\mathbf{x}) - y| - \varepsilon & \text{otherwise.} \end{cases}$$

Other loss functions that have been considered (Gunn, 1998) include a quadratic loss function (which is equivalent to ridge regression) and Huber's loss function. However, the ε-insensitive loss function proposed by Vapnik (which is an approximation to Huber's loss function) is the only one of the three that will produce sparseness in the support vectors; for Huber's loss function and the quadratic loss function all the training data points will be support vectors.

6.3 Mathematical representation of support vector regression

The goal of support vector regression is to find a function $f(\mathbf{x})$ that deviates not more than ε from the actually observed target values, y_i, for the entire training data set. In other words, the regression function will accept errors less than ε. The function f also needs to be as flat as possible.

Consider functions of the form $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$. If we want $f(\mathbf{x})$ to be as flat as possible, it means that we need $\|\mathbf{w}\|$ as small as possible. In other words, we need to

minimise $\|\mathbf{w}\|$, or for convenience sake, to minimise $\frac{1}{2}\|\mathbf{w}\|^2$. This minimisation needs to be done subject to the constraints

$$y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b \leq \varepsilon; \ \langle \mathbf{w}, \mathbf{x}_i \rangle + b - y_i \leq \varepsilon. \tag{3.13}$$

However, a function f which approximates the data with ε precision does not always exist. In such a case, we need to allow for errors. The approach in this instance is similar to the "soft margin" approach taken in the non-linear case for support vector classification. In other words, we introduce slack variables ξ_i, ξ_i^*. The resulting optimisation problem becomes

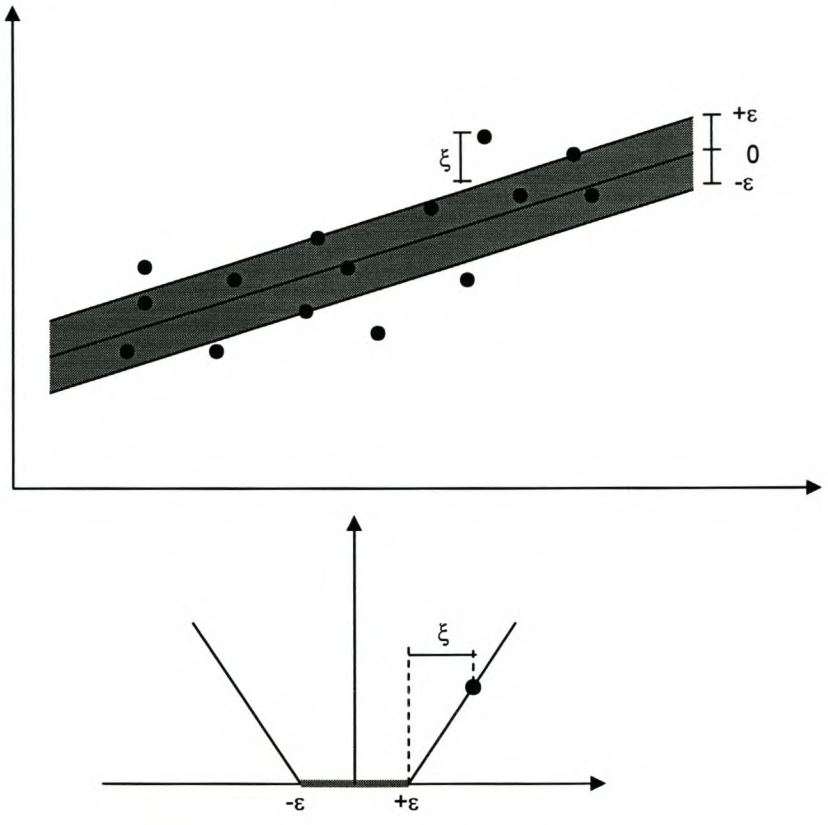
$$\text{minimise } \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

subject to

$$y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b \leq \varepsilon + \xi_i; \qquad \langle \mathbf{w}, \mathbf{x}_i \rangle + b - y_i \leq \varepsilon + \xi_i^*; \quad \xi_i, \xi_i^* \geq 0.$$

The parameter C determines the trade-off between the flatness of the function $f(\mathbf{x})$ and the extent to which deviations greater than ε are tolerated.

Graphically, ε-insensitive support vector regression can be illustrated as follows:



The deviations of the points outside the shaded region are larger than ε, and therefore these points are penalised accordingly. Therefore only these points contribute to the cost. If all the data points lie within or on the $\varepsilon-$insensitive region around the solution, there will be zero error associated with the loss function. The points that are not strictly inside the $\varepsilon-$insensitive tube are support vectors.

As in the case of the SVM for classification, it is easier to solve the dual form of the optimisation problem. To this end, we introduce the Lagrangian

$$L=\frac{1}{2}\|\mathbf{w}\|^2+C\sum_{i=1}^{N}(\xi_i+\xi_i^*)-\sum_{i=1}^{N}\alpha_i(\varepsilon+\xi_i-y_i+\langle\mathbf{w},\mathbf{x}_i\rangle+b)-\sum_{i=1}^{N}\alpha_i^*(\varepsilon+\xi_i^*+y_i-\langle\mathbf{w},\mathbf{x}_i\rangle-b)-\sum_{i=1}^{N}(\eta_i\xi_i+\eta_i^*\xi_i^*)$$

which has to be maximised subject to the positivity constraints $\alpha_i, \alpha_i^*, \eta_i, \eta_i^* \geq 0$. To optimise this function we need to take partial derivatives with respect to $\mathbf{w}, b, \xi_i, \xi_i^*$ and set these partial derivatives equal to zero. This yields

$$\begin{aligned}
\partial_b L &= \sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0 \\
\partial_{\mathbf{w}} L &= \mathbf{w} - \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\mathbf{x}_i = \mathbf{0} \qquad (3.14) \\
\partial_{\xi_i} L &= C - \alpha_i - \eta_i = 0 \\
\partial_{\xi_i^*} L &= C - \alpha_i^* - \eta_i^* = 0.
\end{aligned}$$

Substituting these partial derivatives into the expression for L gives the dual problem of maximising

$$-\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \mathbf{x}_i, \mathbf{x}_j \rangle - \varepsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*)$$

subject to the constraints $\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0$ and $0 \leq \alpha_i, \alpha_i^* \leq C$. From (3.14) we get

$$\mathbf{w} = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\mathbf{x}_i,$$

from which it follows that the fitted regression hyperplane is given by

$$f(\mathbf{x}) = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\langle \mathbf{x}_i, \mathbf{x} \rangle + b.$$

Note that once again the $\mathbf{x}$ – vectors appear in this expression only in the form of an inner product. This makes it easy to extend support vector regression to the non-

linear case: we simply replace every occurrence of an inner product by an appropriate kernel function.

To calculate b, the Karush-Kuhn-Tucker (KKT) conditions must again be introduced. In this case, the relevant KKT conditions are:

$$\alpha_i (\varepsilon + \xi_i - y_i + \langle \mathbf{w}, \mathbf{x}_i \rangle + b) = 0$$

$$\alpha_i^* (\varepsilon + \xi_i^* + y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b) = 0$$

$$(C - \alpha_i)\xi_i = 0$$

$$(C - \alpha_i^*)\xi_i^* = 0 .$$

The KKT conditions imply that only data pairs $(\mathbf{x}_i, y_i)$ for which the corresponding $\alpha_i, \alpha_i^* = C$, lie outside the $\varepsilon-$insensitive tube. Also, $\alpha_i \alpha_i^* = 0$, which means that these dual variables cannot simultaneously be non-zero. Lastly, $0 \le \alpha_i, \alpha_i^* \le C$, which in turn implies that $\xi_i \xi_i^* = 0$.

From the above we see that b can be calculated as

$$b = y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - \varepsilon \qquad \text{for } 0 < \alpha_i < C$$

$$b = y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle + \varepsilon \qquad \text{for } 0 < \alpha_i^* < C.$$

The Lagrange multipliers are non-zero only for $|f(\mathbf{x}_i) - y_i| \geq \varepsilon$, which means that all data pairs inside the ε – insensitive tube have α_i, α_i^* zero. It is said that $\mathbf{w}$ has a sparse expansion in terms of the $\mathbf{x}_i$ (because we do not need all of the $\mathbf{x}_i$ to describe $\mathbf{w}$). The data pairs for which $|f(\mathbf{x}_i) - y_i| \geq \varepsilon$, or where exactly one of the Lagrange multipliers is greater than zero, are called the support vectors.

If $\varepsilon = 0$, the ε – insensitive loss function reduces to the L_1 loss function. Here the optimisation problem is reduced to minimising with respect to $\boldsymbol{\beta}$

$$\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \beta_i \beta_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{i=1}^{N} \beta_i y_i$$

subject to the constraints $-C \leq \beta_i \leq C$, $i = 1, 2, \ldots, N$, and $\sum_{i=1}^{N} \beta_i = 0$. The solutions obtained for $\mathbf{w}$ and b are then given by $\mathbf{w} = \sum_{i=1}^{N} \beta_i \mathbf{x}_i$ and $b = -\frac{1}{2} \langle \mathbf{w}, \mathbf{x}_r \rangle$.

If a quadratic loss function is used, the optimisation problem is to minimise with respect to $\boldsymbol{\beta}$

$$\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \beta_i \beta_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{i=1}^{N} \beta_i y_i + \frac{1}{2C} \sum_{i=1}^{N} \beta_i^2$$

subject to the constraint $\sum_{i=1}^{N} \beta_i = 0$. For a Huber loss function the optimisation problem is to minimise with respect to $\boldsymbol{\beta}$

$$\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \beta_i \beta_j \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle - \sum_{i=1}^{N} \beta_i y_i + \frac{1}{2C} \sum_{i=1}^{N} \beta_i^2 \mu$$

subject to the constraints $-C \leq \beta_i \leq C$, $i = 1, 2, \ldots, N$ and $\sum_{i=1}^{N} \beta_i = 0$.

As with the choice of other parameters, the choice of loss function should be based on prior knowledge of the problem as well as the distribution of the noise. If this information is unknown, Huber's robust loss function might be a good alternative (Vapnik, 1995). Gunn (1998) further states that the ε – insensitive loss function might be an inappropriate choice for certain kernels causing the solution to oscillate within the ε – insensitive region.

A variation on the basic ε – insensitive regression is obtained by modifying the algorithm in such a way that ε need not be specified a priori. Instead, a parameter ν is specified, which is an upper bound on the fraction of errors (i.e. points allowed to lie outside of the tube), and a lower bound on the fraction of support vectors. The corresponding ε is then calculated automatically. This approach is called ν – SV regression. For more detail, see Smola and Schölkopf (2001).

7. Discussion

7.1 Advantages of SVMs

SVMs have been shown to have many advantages over comparable methods:

- The SVM methodology is very flexible. The general binary classification algorithm for linearly separable data can be customised for binary classification on non-linearly separable data (using the concept of duality and the kernel trick), for multi-class classification (either by considering many binary problems or by adjusting the underlying algorithm), as well as regression. Other applications not discussed for which SVMs can also be used, are operator inversion, unsupervised learning, novelty detection, and density estimation.
- Because the quadratic programming problems solved in the solution of the SVM are convex, there are no problems with local minima (since any local minimum found can be identified as the global minimum).
- There are only a few model parameters to pick. Typically, one would only need to pick the kernel function (and its associated parameter/s), as well as the cost parameter.
- Results are easily reproducible. If the same SVM model (with the same parameters) is applied to the same data set twice, the same solution (except for some numeric issues) will be obtained. With neural networks, for instance, results are dependent on not only the particular algorithm used, but also the starting point used.

- SVMs have been proven to be robust to noise.
- SVMs provide generalisation control through a technique that addresses the curse of dimensionality. Other models such as traditional neural network approaches often produce models that can overfit the data, due to the optimisation algorithms used for parameter selection and the statistical measures used to select the 'best' model.

7.2 Some issues that still need to be addressed

- Parameter choice: SVMs may be very sensitive to the proper choice of parameter values (Meyer, 2002).
- Do SVMs scale to massive data sets? The optimisation algorithms used for solving SVM models are robust, and results have been reported for classification problems with millions of data cases. However, the problem remains that the size of the quadratic programming problem scales with the number of support vectors. The problem can be solved through chunking, in which the support vectors are extracted by going through the training data in chunks, while regularly testing for the possibility that patterns initially not identified as support vectors become support vectors at a later stage. Without this procedure, the size of the matrix in the quadratic part of the objective function would be $N \text{ x } N$, where N is the number of training examples.
- Currently SVMs can only incorporate prior knowledge through the preparation of the data and the choice of kernel.
- Data formatting issues such as the handling of missing values and the treatment of categorical variables need to be addressed.

7.3 Applications of SVMs

SVMs have been successfully applied in a variety of fields, including:

- Optical character recognition
- Particle identification
- Face identification / image classification
- Text categorisation
- Engine knock detection
- Database marketing
- Hand-written character recognition
- Bioinformatics (including protein homology detection, gene expression)
- Function approximation
- Cancer diagnosis
- High-energy physics
- Forecasting time series
- Novelty detection
- Credit scoring

Chapter 4

The computing environment

1. Tools for implementing traditional techniques

Today there are many sophisticated software packages available for data analysis. Some of the most well-known are SAS, SPSS and Statistica. Most packages consist of a base package together with some add-on component for data analysis. For instance, the data mining tool of SAS is known as Enterprise Miner, while Statistica has an add-on component called Data Miner. These all function in a menu-driven "point-and-click" environment. Packages such as S-Plus, which can actually be described as a statistical programming language, are also available. In all of these packages, most classical and modern techniques are readily available. For instance, Enterprise Miner includes decision trees, neural networks, regression, memory-based reasoning, bagging and boosting ensembles, two-stage models, clustering, time series, and associations. (http://www.sas.com/technologies/analytics/datamining/miner/).

Statistica Data Miner includes five general techniques, namely a general slicer / dicer and drill-down explorer, general classifier, general modeller / multivariate explorer,

general forecaster, and general neural networks explorer. It also comes with more specialist techniques such as generalised additive models, mining for association rules, feature selection and variable filtering (for very large datasets), general classification and regression trees, general CHAID models, boosted tree classifiers and regression, interactive trees, MAR splines, naive Bayes classifiers, and k-nearest neighbour techniques. (http://www.statsoft.com/Dataminer.html).

2. SVM implementation

Compared to the techniques mentioned above, SVMs are somewhat newer, and have until very recently not yet been incorporated into most standard data mining packages. Statistica recently added a support vector machine routine to its Data Miner; details can be found at http://www.statsoftinc.com/uk/datamin.html. SAS has also recently announced that support vector machines will be included in its new version of Enterprise Miner that will be released together with SAS 9.1 early in 2004 (Milley, 2003).

There are, however, also many other possibilities for implementing SVMs that have been used up to now because of the lack of routines in standard commercial packages. Some of the most well-known packages available include LIBSVM, SVMLight and SVMTorch. Steve Gunn's SVM Matlab toolbox is also popular (see Gunn, 1998).

LIBSVM was created by Chih-Chung Chang and Chih-Jen Lin, from the National Taiwan University. They list the main features of LIBSVM as:

- different SVM formulations
- efficient multi-class classification
- cross-validation for model selection
- weighted SVM for unbalanced data
- both C++ and Java sources
- GUI demonstrating SVM classification and regression
- Python, R, SPlus, Matlab, Perl and Ruby interfaces
- Automatic model selection which can generate contours of cross-validation accuracy

LIBSVM can be downloaded from http://www.csie.ntu.edu.tw/~cjlin/libsvm.

The author of SVMLight is Thorsten Joachims, from the University of Dortmund. He lists the main features of SVMLight as:

- fast optimisation algorithm
- solves classification, regression and training problems
- computes XiAlpha estimates of the error rate, the precision and the recall
- efficiently computes leave-one-out estimates of the error rate, the precision and the recall
- includes algorithms for approximately training large transductive SVMs
- can train SVMs with cost models
- handles many thousands of support vectors
- handles several hundred thousands of training examples
- supports standard kernel functions and lets you define your own
- uses sparse vector representation

SVM Light is available from http://svmlight.joachims.org.

SVMTorch is the SVM component of the Torch Machine learning library from the Dalle Molle Institute for Perceptual Artificial Intelligence. It is billed as an SVM specifically for large-scale regression and classification problems, such as problems with more than 20 000 examples, and even for input dimensions higher than 100. For more details on SVMTorch, see Collobert and Bengio (2001).

A list of these and other software available for the implementation of support vector machines can be found at http://www.kernel-project.org/software.

3. The R package

As can be seen from the above, there are many different software packages available for implementing the different techniques that will be compared. For ease of comparison, it was decided to use a single software package in the final analysis, and the software that was decided on is the R package.

R is a computing language and environment that can be used for statistical computing and graphics. It is very similar to the award-winning S-system, which was developed at Bell Laboratories by John Chambers et al. in the 1980's, and is widely used in the statistical community; in fact, R is considered to be a dialect of S. The big difference, however, is that R is free software that can be downloaded directly from the web.

R is a project of the Free Software Foundation. It was originally developed by Robert Gentleman and Ross Ihaka from the Statistics Department at the University of Auckland. It is currently maintained and updated by a core development group, whose members include John Chambers, Brian Ripley and about 15 others, but it is basically the result of contributions from many individuals. It is available via CRAN (Comprehensive R Archive Network) which has many mirror sites all over the world, and versions of R are available for most operating systems.

R offers, amongst other things

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either directly at the computer or on hardcopy, and
- a well developed, simple and effective programming language.

R is an environment in which many classical and modern statistical techniques have been implemented. Some of these have been built into the base R environment, but many are supplied as packages. There are about 8 standard packages that are supplied with R, and many more that are available through CRAN. Most classical statistics techniques and much of the latest methodology are available for use with R, but users will need to be prepared to do a little work to find it. The contents of some of the packages are quite obvious; for example, classification and regression trees are available in a package called "tree", and feed-forward neural networks in a package

called "nnet". Certain standard techniques such as ordinary regression (fitting linear models) are available in the base package of R. However, linear discriminant analysis is found in the "MASS" package, which is a library developed by Venables and Ripley to accompany their standard book (see Venables and Ripley, 1994). Support vector machines can be found in the "e1071" package, which contains miscellaneous functions of the Department of Statistics at the Technische Universität Wien, Austria. Together with the functions for SVM implementation, this package also includes functions for techniques such as latent class analysis (LCA), bagged clustering and fuzzy C-means clustering.

The author of the SVM routine in e1071 is David Meyer of the Technische Universität Wien, and it is based on the award-winning C++ implementation LIBSVM by Chih-Chung Chang and Chih-Jen Lin.

Further information on R can be found on the web at http://www.r-project.org.

Chapter 5

Classification problem

1. Description of the basic problem

In the life assurance industry – as in most industries – client retention is very important. One aspect of client retention in this industry, is the issue of early policy termination; in other words, lapses and surrenders. Lapses and surrenders both occur when a client terminates a policy before the policy has run its term; however, there is a distinct difference. From a client perspective, a termination is considered as a lapse when no money is paid out to the client upon termination, while some form of payout of accumulated value is made to the client in the case of a surrender. From a company perspective, a termination during the initial period of the policy is considered to be a lapse, while a surrender is a termination after the initial period. The initial period can be different for different product types. In essence, however, a lapse usually means that there is a financial loss for the company, as the policy is terminated before the period in which costs are recouped has elapsed.

Determining or predicting whether a client will lapse a policy is therefore very important for profitability, and it is thus crucial to be able to predict beforehand whether a client is likely to lapse a policy.

Currently Sanlam has a lapse probability model called the 'odds' model. This model was developed externally for Sanlam, and calculates the probability of a policy lapse within the first twelve months of inception. Every new policy application is evaluated on the basis of the model, and the risk value of a potential policy is calculated pro-actively, enabling the information to be used as a management instrument.

However, currently no model exists that works on the entire client database. Such a model is needed to determine which policies are at great risk of lapsing in order to manage this. It would be especially beneficial to assist with client retention projects. An additional benefit of such a model could be its use as an aid in client selection for direct mail projects: currently, credit scoring is sometimes used to enhance client selection. However, credit scoring can be rather expensive, as a score has to be purchased for each client from external providers. Although credit scoring can be very useful in campaigns targeted at non-clients, in the case of campaigns targeted at existing clients it is hypothesised that Sanlam probably has more relevant information on its own clients than a credit bureau could possibly have. A model such as a lapse model could therefore probably be more successful in a pre-screening process than credit scores could be.

In this chapter we report on an empirical study that was undertaken with the aim of developing a model which can be used to classify a client into one of two categories,

viz. the client should be considered at risk of lapsing a policy, or not. Data from the Sanlam database were used in the study. As a starting point, it was decided that the problem would be considered from a client perspective instead of a policy perspective. In other words, instead of determining whether a specific policy is likely to lapse, the goal was only to determine whether a client was likely to lapse any policy, i.e. whether a client had the typical profile of a person who would lapse a policy.

2. Definition of the input and response variables

In the Sanlam database records are maintained separately for policyholders (individuals) and policies, since there can be a one-to-many link between these records. Not all data were extracted for purposes of this study, as information such as underwriting information was not deemed relevant for the purposes of this analysis. The aim was to model lapse information; therefore, a random sample of clients who had lapsed a policy between February and August 2003 was drawn. For these clients, the month in which the lapse took place was determined, and historical client data for the month preceding the lapse were then extracted. This was done in order to get a snapshot of what the client's data looked like just before the policy had lapsed. A random sample of non-lapse cases was also selected, and for these cases the current client data were extracted.

All personal data fields were extracted, as well as aggregated policy information. In other words, if an individual had more than one policy, all the policy data were combined into a single record.

The response (dependent) variable was chosen as an indicator of whether the person had lapsed a policy or not: the field name is "lapse" and has values of 0 or 1, where 1 means that the person has at least one lapsed policy.

The input (independent) variables consisted of all the other available client personal and aggregated policy information.

3. Data cleaning

Initial data cleaning entailed eliminating duplicated information - this happened where certain variables were encoded in other variables. For example, there was a variable for the total number of policies, and also for the total number of recurring policies and the total number of single premium policies. The number of policies was also further broken down into the number of policies per each product category, and even further, the number of policies per each individual product type. Obviously it was unnecessary to have individual fields as well as different combinations of sums of these fields; therefore, most fields that could be deduced in this way were either deleted or the aggregates were used. The same problem also occurred with the premium information, as well as the time since last purchase information.

Categorical variables were also transformed to be contained in separate fields. For instance, an original gender variable was coded as 1 for male and 2 for female. This was transformed to a binary (0 / 1) variable with a 1 indicating a male. Similar transformations were performed for race, title, marital status, language, province and purchase channel (i.e. whether a client purchased his / her last product from a company representative, an independent broker, or directly).

4. Descriptive measures for the data

After all of the above data cleaning, the dataset consisted of 9 596 cases, of which 4 851 were lapse cases, and 4 745 were non-lapse cases. There were 61 input (independent) variables and 1 response (dependent) variable (i.e. the lapse indicator). Of the 61 input variables, 36 were categorical and 25 numerical. The response variable was categorical.

On the following pages, some descriptive measures for the data are given. Firstly, a table with descriptions of each of the variables is given, followed by a table with descriptive statistics for the data (mean, median, standard deviation and range). The correlation of each input variable with the response variable is also given.

Variable name	Variable description	Variable type
Lapse	**Flag indicating whether the client has had a lapsed policy (RESPONSE VARIABLE)**	**Categorical**
GEND_MALE	Gender indicator	Categorical
TITLE_ORD	Flag indicating an ordinary title (Mr. / Mrs. / Miss / Ms)	Categorical
TITLE_PROF	Flag indicating a professional title (e.g. doctor, advocate, etc.)	Categorical
TITLE_CHURCH	Flag indicating a religious title (e.g. reverend)	Categorical
TOTAL_INC	Total income of client (own + spouse's income) [outdated in some cases]	Numerical
PERS_FAMILY_SZ	Family size of client [outdated in many cases]	Numerical
MRTL_ST_SGL_NM	Flag indicating whether marital status is single (never been married)	Categorical
MRTL_ST_MAR	Flag indicating whether marital status is married / living together	Categorical
MRTL_ST_SGL_WM	Flag indicating whether marital status is divorced / widowed	Categorical
PERS_PPS_FLG	Flag indicating whether client is also a client of the Professional Provident Society	Categorical
LANG_AFR	Preferred language of client (1 = Afrikaans; 0 = English)	Categorical
PSTL_ADR_UNKWN_FLG	Flag indicating whether client's postal address is unknown	Categorical
CHAN_ADV	Flag indicating whether client has an adviser (company representative) as intermediary	Categorical
CHAN_BROK	Flag indicating whether a client has a broker as intermediary	Categorical
CHAN_ORPH	Flag indicating whether client is orphaned (i.e. has no intermediary on record)	Categorical
AGE	Age (in years)	Numerical
RACE_WHITE	Flag indicating whether client's race is white	Categorical
RACE_BLACK	Flag indicating whether client's race is black	Categorical
RACE_COLRD	Flag indicating whether client's race is coloured	Categorical
RACE_ASIAN	Flag indicating whether client's race is Asian	Categorical
PROV_GAUTENG	Flag indicating whether client resides in Gauteng	Categorical
PROV_WC	Flag indicating whether client resides in the Western Cape	Categorical
PROV_KZN	Flag indicating whether client resides in Kwazulu Natal	Categorical
PROV_EC	Flag indicating whether client resides in the Eastern Cape	Categorical
PROV_FS	Flag indicating whether client resides in the Free State	Categorical
PROV_LP	Flag indicating whether client resides in Limpopo	Categorical
PROV_MP	Flag indicating whether client resides in Mpumalanga	Categorical
PROV_NW	Flag indicating whether client resides in North West	Categorical
PROV_NC	Flag indicating whether client resides in the Northern Cape	Categorical
ann_recur_prem	Total annual recurring premiums from client	Numerical
sngl_prems	Total single premiums from client	Numerical
Monthly	Flag indicating whether client has any products that are paid on a monthly basis	Categorical
paid_up	Flag indicating whether client has any paid-up products	Categorical
time_W_S	Number of months has been a client of Sanlam	Numerical
time_R_S	Number of months client has remaining with Sanlam (longest remaining term on all policies)	Numerical
child_endow	Number of child endowment products	Numerical
Endowm	Number of endowment products	Numerical
Funeral	Number of funeral insurance products	Numerical
life_ann	Number of life annuity products	Numerical
OLV	Number of OLV products	Numerical
Ra	Number of retirement annuities	Numerical
snkng_fnd	Number of sinking fund products	Numerical
trm_ann	Number of term annuity products	Numerical
w_life	Number of whole life products	Numerical
Matrix	Number of Matrix products	Numerical
Ut	Number of unit trust products	Numerical
T_LPOL	Time in months since client's last Sanlam product purchase	Numerical
had_mat	Flag indicating whether client has had a matured policy	Categorical

Variable name	Variable description	Variable type
tot_cov	Aggregate of sum of cover of all active products	Numerical
tot_cov2	Maximum of all active products' cover	Numerical
deb_ord	Flag indicating whether client pays premiums by debit order	Categorical
PLAN_CNT	Number of plans	Numerical
no_r_agmts	Number of recurring products	Numerical
no_s_agmts	Number of single premium products	Numerical
ave_time_between	Average time between Sanlam product purchases	Numerical
LOAN_FLG_YES	Flag indicating whether it is known that a client has an outstanding loan against a policy	Categorical
LOAN_FLG_NO	Flag indicating whether it is known that a client does not have an outstanding loan against a policy	Categorical
LOAN_FLG_UNK	Flag indicating that it is unknown whether a client has an outstanding loan against a policy	Categorical
PREM_DEBT_FLG_YES	Flag indicating whether it is known that a client has premium debt against a policy	Categorical
PREM_DEBT_FLG_NO	Flag indicating whether it is known that a client does not have premium debt against a policy	Categorical
PREM_DEBT_FLG_UNK	Flag indicating that it is unknown whether a client has premium debt against a policy	Categorical

Variable name	Mean / mode[1]	Median[2]	SD[3]	Range		Corr.[4]
				Min	Max	
Lapse	**1**		**0.500**	**0**	**1**	**1**
GEND_MALE	1		0.498	0	1	0.013
TITLE_ORD	1		0.154	0	1	0.017
TITLE_PROF	0		0.139	0	1	-0.008
TITLE_CHURCH	0		0.051	0	1	-0.011
TOTAL_INC	86046	58000	98629	0	1440000	0.115
PERS_FAMILY_SZ	1.439	2	0.773	0	10	0.117
MRTL_ST_SGL_NM	0		0.447	0	1	-0.029
MRTL_ST_MAR	1		0.491	0	1	0.033
MRTL_ST_SGL_WM	0		0.255	0	1	0.072
PERS_PPS_FLG	0		0.248	0	1	-0.016
LANG_AFR	1		0.498	0	1	0.032
PSTL_ADR_UNKWN_FLG	0		0.201	0	1	-0.049
CHAN_ADV	0		0.489	0	1	0.091
CHAN_BROK	0		0.483	0	1	0.005
CHAN_ORPH	0		0.402	0	1	-0.176
AGE	43	41	12.979	4	94	0.014
RACE_WHITE	1		0.472	0	1	0.009
RACE_BLACK	0		0.423	0	1	0.014
RACE_COLRD	0		0.253	0	1	-0.026
RACE_ASIAN	0		0.180	0	1	-0.021
PROV_GAUTENG	0		0.478	0	1	0.011
PROV_WC	0		0.390	0	1	0.003
PROV_KZN	0		0.334	0	1	-0.019
PROV_EC	0		0.300	0	1	0.044
PROV_FS	0		0.246	0	1	0.007
PROV_LP	0		0.151	0	1	-0.012
PROV_MP	0		0.185	0	1	-0.016
PROV_NW	0		0.223	0	1	0.005
PROV_NC	0		0.175	0	1	-0.014
ann_recur_prem	6666	3840	11471	0	265824	0.187
sngl_prems	13957	0	56097	0	1858991	0.161
Monthly	1		0.323	0	1	0.180
paid_up	0		0.363	0	1	-0.013
time_W_S	122	98	88	0	864	-0.035
time_R_S	4170	342	4795	0	9999	0.181
child_endow	0.103	0	0.384	0	5	0.038
Endowm	0.835	1	1.006	0	19	0.193
Funeral	0.108	0	0.325	0	2	0.091
life_ann	0.101	0	0.368	0	8	0.222
OLV	0.200	0	0.519	0	3	0.328
Ra	0.717	1	0.865	0	12	0.111
snkng_fnd	0.022	0	0.183	0	7	0.067
trm_ann	0.073	0	0.335	0	8	0.164
w_life	0.292	0	0.632	0	10	0.117
Matrix	0.028	0	0.183	0	4	0.072
Ut	0.159	0	0.528	0	8	0.079
T_LPOL	50	34	58	-3	728	-0.444
had_mat	0		0.298	0	1	0.082
tot_cov	333484	134401	635454	0	9999999	0.184
tot_cov2	199081	100000	434819	0	9999999	0.180
deb_ord	1		0.465	0	1	0.191
PLAN_CNT	0.167	0	0.490	0	10	0.266

				Range		
no_r_agmts	2.125	2	1.677	0	19	0.378
no_s_agmts	0.263	0	0.728	0	13	0.215
ave_time_between	30	21.5	36	0	667	0.092
LOAN_FLG_YES	0		0.057	0	1	-0.013
LOAN_FLG_NO	0		0.231	0	1	0.008
LOAN_FLG_UNK	1		0.237	0	1	-0.004
PREM_DEBT_FLG_YES	0		0.054	0	1	-0.020
PREM_DEBT_FLG_NO	0		0.232	0	1	0.009
PREM_DEBT_FLG_UNK	1		0.237	0	1	-0.004

1 For numerical (continuous) variables the mean is given; for categorical variables the mode
2 The median is only given for numerical variables
3 Standard deviation of the variable
4 Correlation of the variable with the response variable "lapse"

5. Selection of variables

During initial experiments it was observed that the different procedures seemed to perform better if only a subset of the available input variables, rather than the full set of 61, was used. This is not unexpected: it is a well known fact that the performance of many statistical procedures can be improved significantly by using an appropriate subset of the input variables. However, there is no procedure in R for automatic selection of variables in a support vector machine. A few variable sets were therefore constructed using different ad hoc approaches.

First, an extremely oversimplified approach to variable selection was followed: the data were read into SAS Enterprise Miner, and basic logistic regression models and basic classification tree models were built. Six variations of the models were built: 4 logistic regression models and 2 trees. For the 4 regression models 4 different variable selection methods were used: stepwise, backward, forward (all using Akaike's Information Criterion with a significance level of 0.05), as well as a backward selection method using validation error as criterion. The 2 tree models consisted of one allowing only binary splits, and one allowing a maximum of 5 branches per node. The variables selected by each of these 6 methods were noted, and it was decided to use only those variables selected by at least 3 of the 6 models. A total of 19 variables (over and above the dependent variable) were selected in this way; they are: TOTAL_INC, MRTL_ST_SGL_NM, MRTL_ST_MAR, MRTL_ST_SGL_WM, AGE, PROV_GAUTENG, ann_recur_prem, paid_up, time_W_S, funeral, OLV, snkng_fnd, trm_ann, T_LPOL, tot_cov, deb_ord, PLAN_CNT, no_r_agmts, ave_time_between (Variable set A).

During the above process it was noted that the tree and logistic regression models tended to select different variables. Another variation was therefore attempted, using only the variables selected by the tree using 5 splits. This model selected the following variables: AGE, RACE_WHITE, paid_up, funeral, OLV, T_LPOL, tot_cov, PLAN_CNT, no_r_agmts (Variable set B).

Another variable set was constructed utilising only those input variables with a correlation of more than 0.2 with the response variables. These variables are: life_ann, OLV, T_LPOL, PLAN_CNT, no_r_agmts, no_s_agmts (Variable set C).

The last variable set consisted of all 61 input variables (Variable set D).

6. Model and parameter selection

Preliminary analysis using a neural network in Enterprise Miner showed that this technique did not perform as well in terms of classification accuracy as logistic regression and trees. Also, the performance of logistic regression was slightly worse than that of classification trees. It was therefore decided that only linear discriminant analysis, support vector machines and classification trees would be compared in the final analysis. An R program was written (see Appendix B) to perform the data analysis.

The data analysis comprised the following steps. The available data were randomly divided into a training set (typically, 75% of the total) and a test set (the remaining

25%). A model was fit to the training data, and then the model was used to predict the lapse category of the data cases in the test data set. The prediction error was calculated as the percentage of misclassifications in the test data set. For each model, the data split into training and test data was performed 100 times, and the average error rate was calculated over the 100 iterations. (By error rate is simply meant the proportion / percentage of misclassified test data cases).

No parameter tuning was performed for the discriminant analysis and tree models; for SVMs, several combinations of parameters were however tried.

The following parameters were varied (over and above the variation of the variable set, as defined in the previous paragraph):

- kernel (the following kernels were tried: a radial basis function [RBF], a second degree polynomial, third degree polynomial and a sigmoidal kernel)
- cost parameter (C) of SVM (initially only values of 1, 50, 100 and 1000 were tried. The best area seemed to be between 1 and 50; therefore a grid search was done including C equal to 10, 20, 30 and 40.)
- gamma parameter (γ) of the RBF kernel (initially only values of 0.1, 0.01, 0.001 and 0.00001 were tried; the best area seemed to be between 0.001 and 0.01, and a grid search was therefore done including γ equal to 0.0025, 0.005 and 0.0075.)

7. Results

The results for linear discriminant analysis (LDA) and classification trees were reasonably constant. For the 37 cases that were run with variable set A, the descriptive statistics for the mean error rates obtained by these two techniques are as follows:

Table 5.1: Descriptive statistics for mean error rates for linear discriminant analysis and classification trees (variable set A)

Technique	Mean	Standard deviation	Range	
			Minimum	Maximum
LDA	0.18225	0.000733	0.180721	0.183592
Tree	0.179097	0.000746	0.17775	0.180575

With variable set B both the classification tree and LDA performed slightly worse (mean error rates of 0.18079 and 0.187191 respectively). Variable sets C and D also performed worse than variable set A. The very small variation in the error rates for the classification trees and the linear discriminant analyses using variable set A suggested that the error rates for the SVMs using different parameters could be compared directly to find the best performing parameter set, since the sample variation appeared to be very small.

In the investigation of different kernels, a second-degree polynomial kernel was firstly compared to an RBF kernel. Variable set A was used. The differing error rates for the two kernels for corresponding cost and gamma parameters are given below:

Table 5.2: Error rates for second-degree polynomial and RBF kernels for different values of cost and gamma (variable set A)

Parameters	ERROR RATES	
	Second-degree polynomial	Radial basis function
C = 1, γ = 0.001	0.164529	0.165904
C = 50, γ = 0.001	0.14795	0.146233
C = 100, γ = 0.001	0.147692	0.148279
C = 1, γ = 0.1	0.152079	0.146983

In some cases the second-degree polynomial fared better, while in other cases the RBF kernel did better. However, the differences were very slight.

In addition, a third-degree polynomial kernel as well as a sigmoidal kernel function were also used to fit a support vector machine to the data. They seemed to do less well than the second-degree polynomial and the radial basis function.

The RBF kernel and the second-degree polynomial therefore seemed to be the most suitable for the data set. However, Meyer (2002) states that the RBF kernel is the only kernel that is currently optimised for use in *R*. The decision was therefore made to use the RBF kernel in further analysis.

Next, the variable sets were varied. In the table below, the different error rates for corresponding parameters fit to variable sets A and B respectively are given:

Table 5.3: Error rates for different cost and gamma values for variable sets A and B

Parameter	ERROR RATE	
	Variable set A	Variable set B
C = 1, γ = 0.001	0.165904	0.169808
C = 50, γ = 0.001	0.146233	0.152658
C = 100, γ = 0.001	0.148279	0.151129
C = 1000, γ = 0.001	0.150058	0.152488
C = 1, γ = 0.01	0.146683	0.151329
C = 50, γ = 0.01	0.147879	0.147808
C = 100, γ = 0.01	0.148208	0.14765
C = 1000, γ = 0.01	0.153796	0.145988

Variable set A seemed to outperform variable set B in most instances. Variable sets C and D were also used, but variable set A consistently outperformed the other variable sets.

Next, the effect of different cost and gamma parameters was examined. An RBF kernel was used together with variable set A. Initially only values of 1, 50, 100 and 1000 were used for C, and values of 0.1, 0.01, 0.001 and 0.00001 for γ. The best performing area for C seemed to be somewhere between 1 and 50, and the best performing area for γ somewhere between 0.001 and 0.01. A grid search for the best combination of C and γ was therefore attempted; for C, values of 10, 20, 30 and 40 were included, while for gamma values of 0.0025, 0.005 and 0.0075 were included.

The error rates for these parameter combinations are given in the table below. It appears that no clear-cut "best" combination exists, but rather that there is a relationship between the optimal values for C and γ. The optimal value of C seems to be somewhere between 1 and 50, while the optimal value for γ appears to be

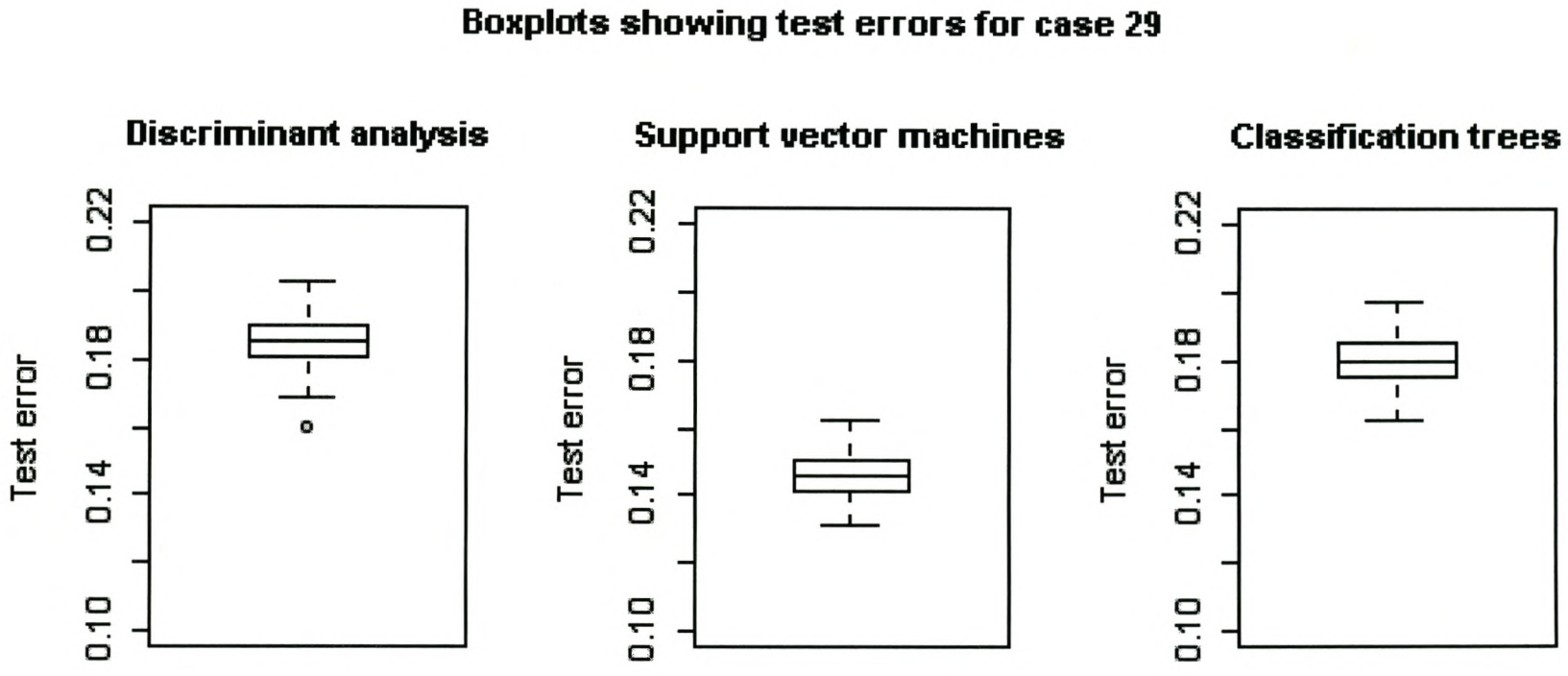
somewhere between 0.001 and 0.1. Larger values of C seem to go together with smaller values of γ, while smaller C values go with larger γ values.

Table 5.4: Error rates for different cost and gamma combinations (variable set A)

C	γ					
	0.001	**0.0025**	**0.005**	**0.0075**	**0.01**	**0.1**
1	0.165904				0.146683	0.146983
10	0.148217	0.146754	0.147825	0.1477	0.147775	
20	0.148125	0.148333	0.149438	0.1489	0.14835	
30	0.148754	0.148758	0.148938	0.147508	0.146567	
40	0.147238	0.149142	0.147888	0.147663	0.147767	
50	0.146233				0.147879	0.172946
100	0.148279				0.148208	
1000	0.150058				0.153796	

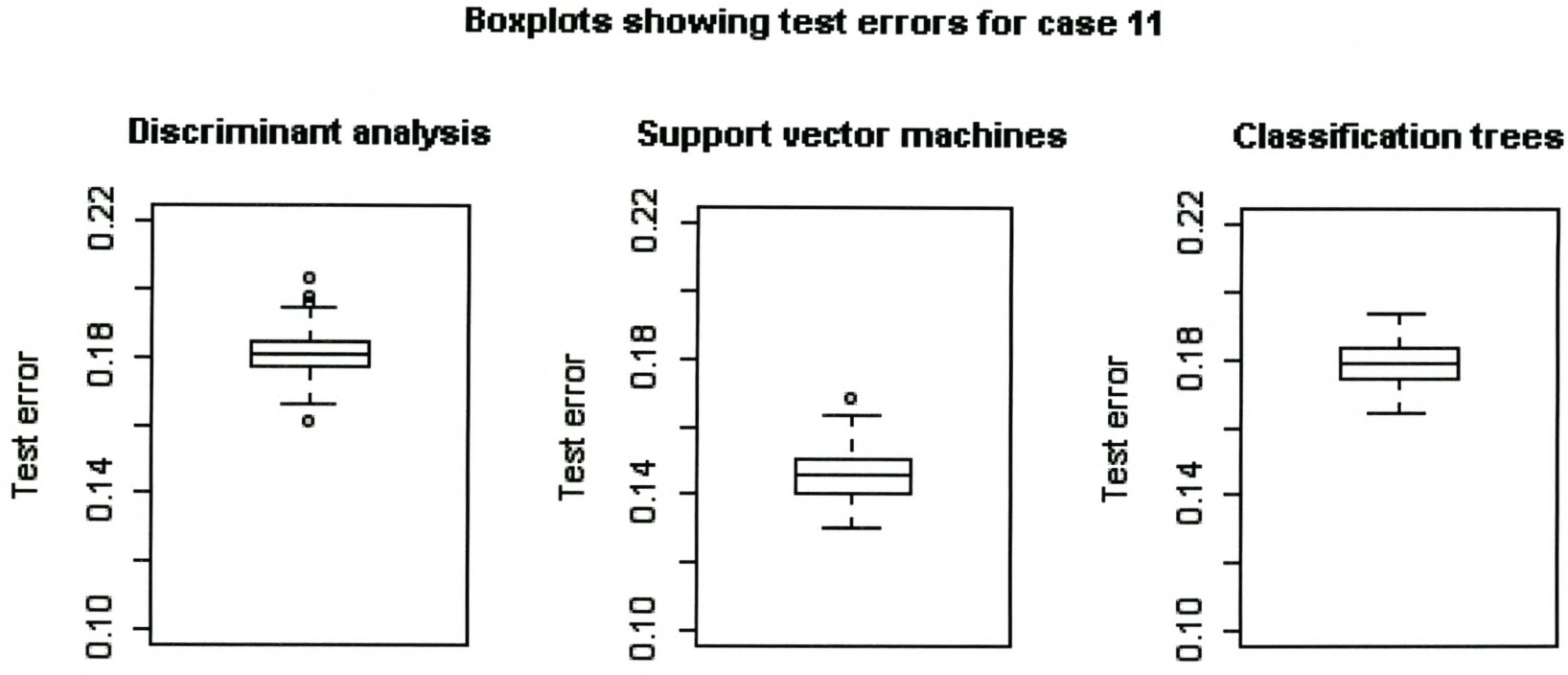
The best performing SVM overall, was the one using variable set B with an RBF kernel and a cost parameter of 1000 together with a γ value of 0.01. However, overall, variable set A seemed to do better than variable set B.

On the following pages, box plots of the error rates for the 10 best performing SVM cases are given, together with the corresponding box plots for linear discriminant analysis and classification trees. It is clear that in all cases the SVM fares better than the other two methods. In fact, with the exception of a few extreme cases, SVM performance was better than that of discriminant analysis and classification trees every time.



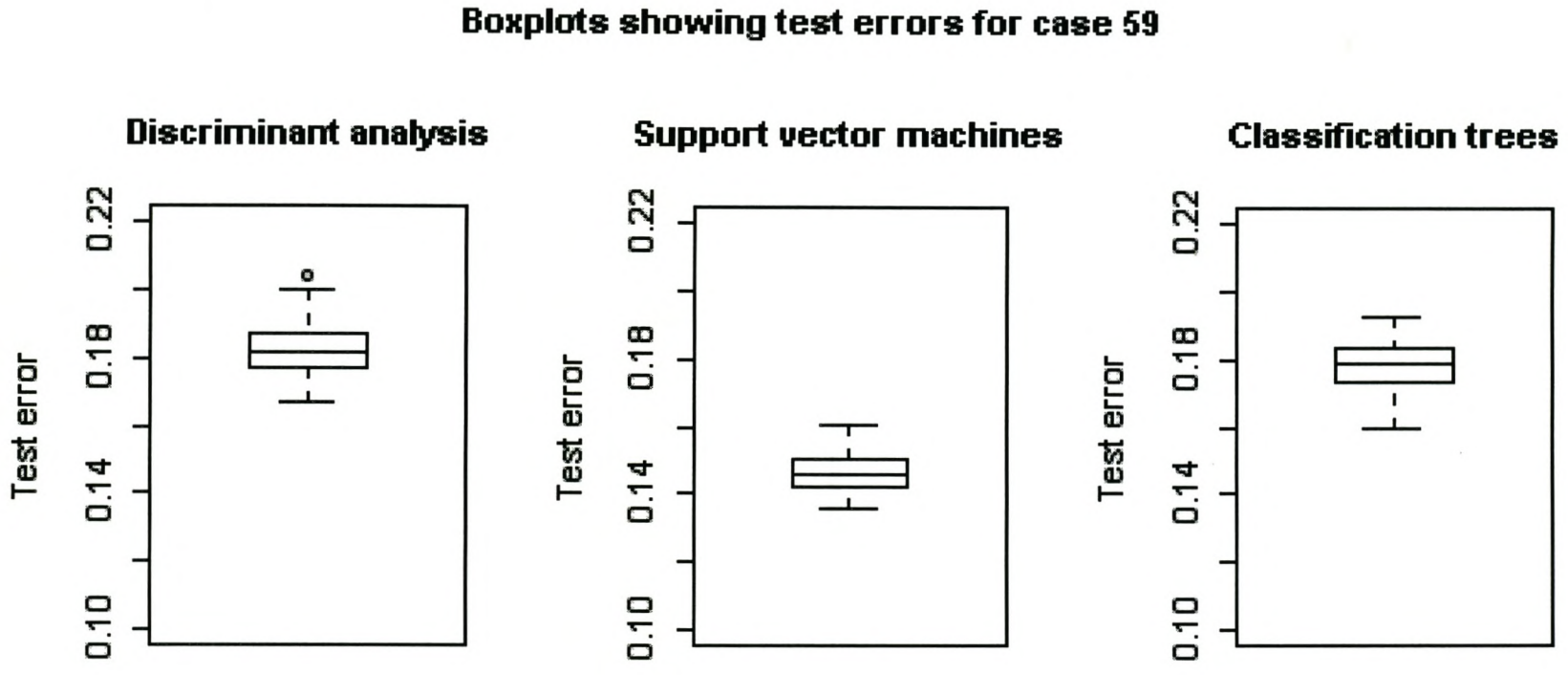
	LDA	SVM	Tree
Mean error rate	0.185408	0.145988	0.179779

CASE 29: Variable set B, RBF kernel, C = 1000, $\gamma = 0.01$



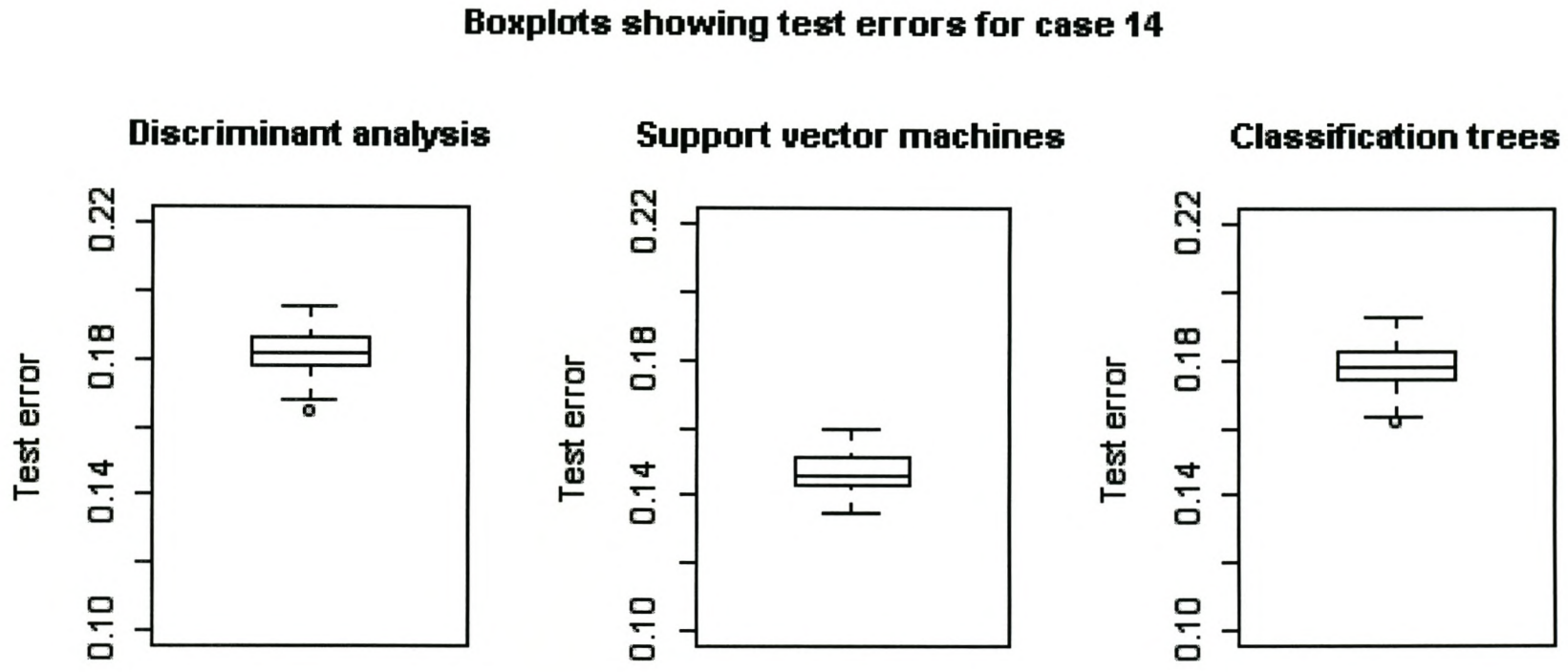
	LDA	SVM	Tree
Mean error rate	0.180721	0.146233	0.178375

CASE 11: Variable set A, RBF kernel, C = 50, $\gamma = 0.001$



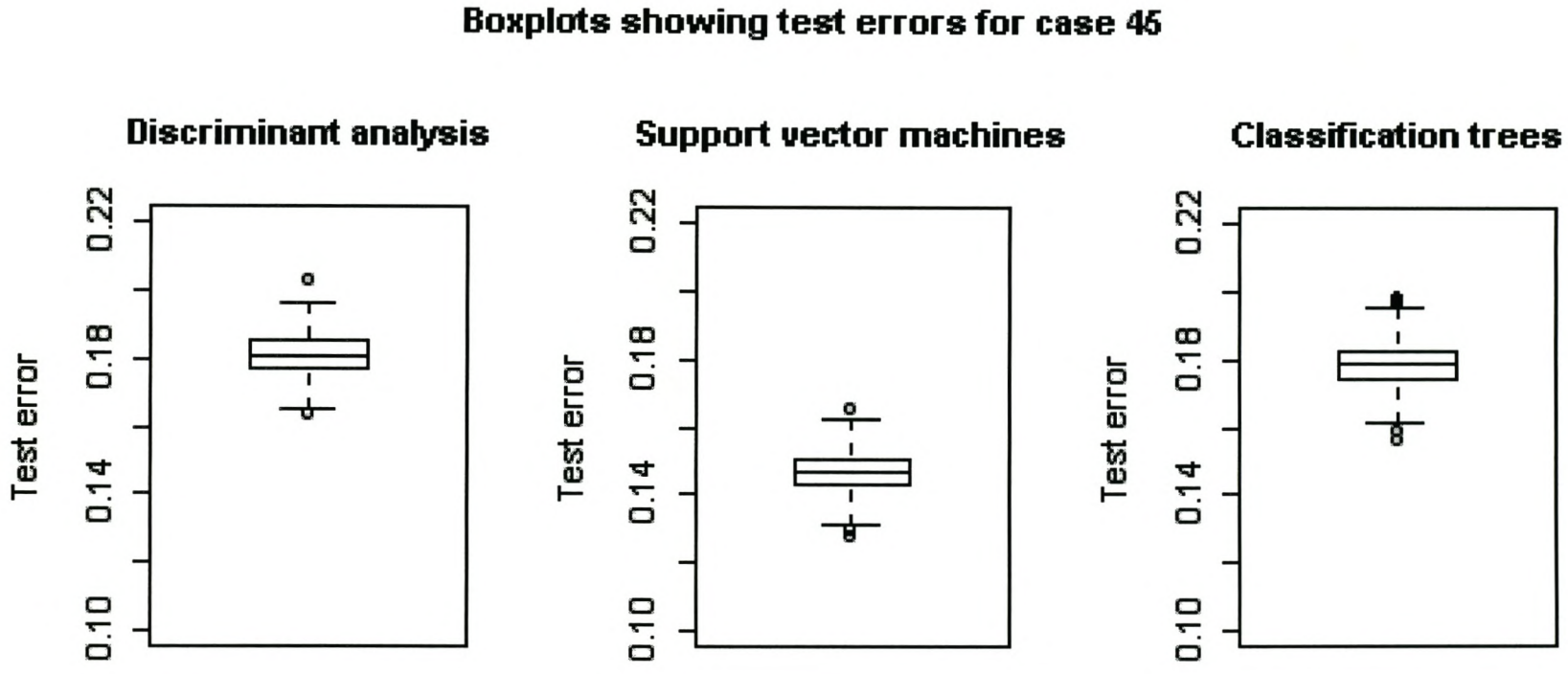
	LDA	SVM	Tree
Mean error rate	0.182267	0.146567	0.178717

CASE 59: Variable set A, RBF kernel, C = 30, $\gamma = 0.01$



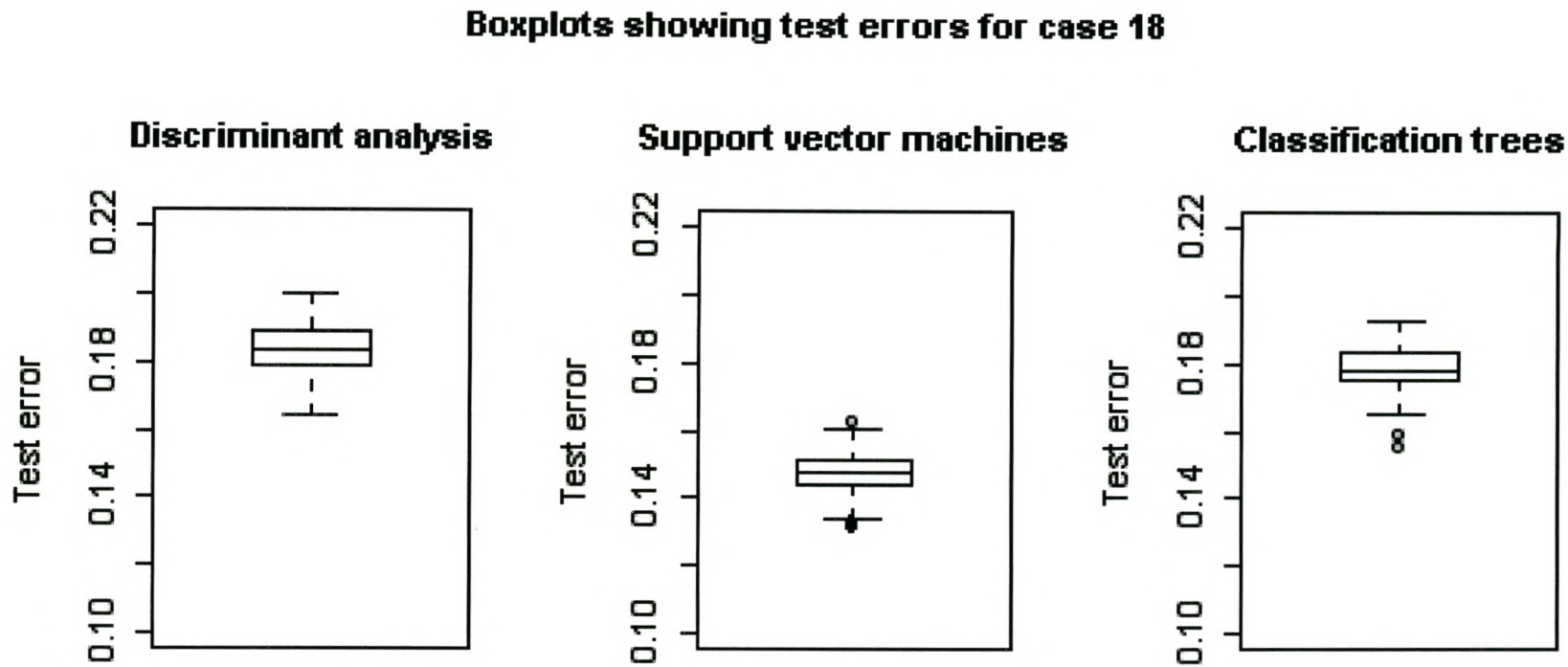
	LDA	SVM	Tree
Mean error rate	0.181867	0.146683	0.177950

CASE 14: Variable set A, RBF kernel, C = 1, γ = 0.01



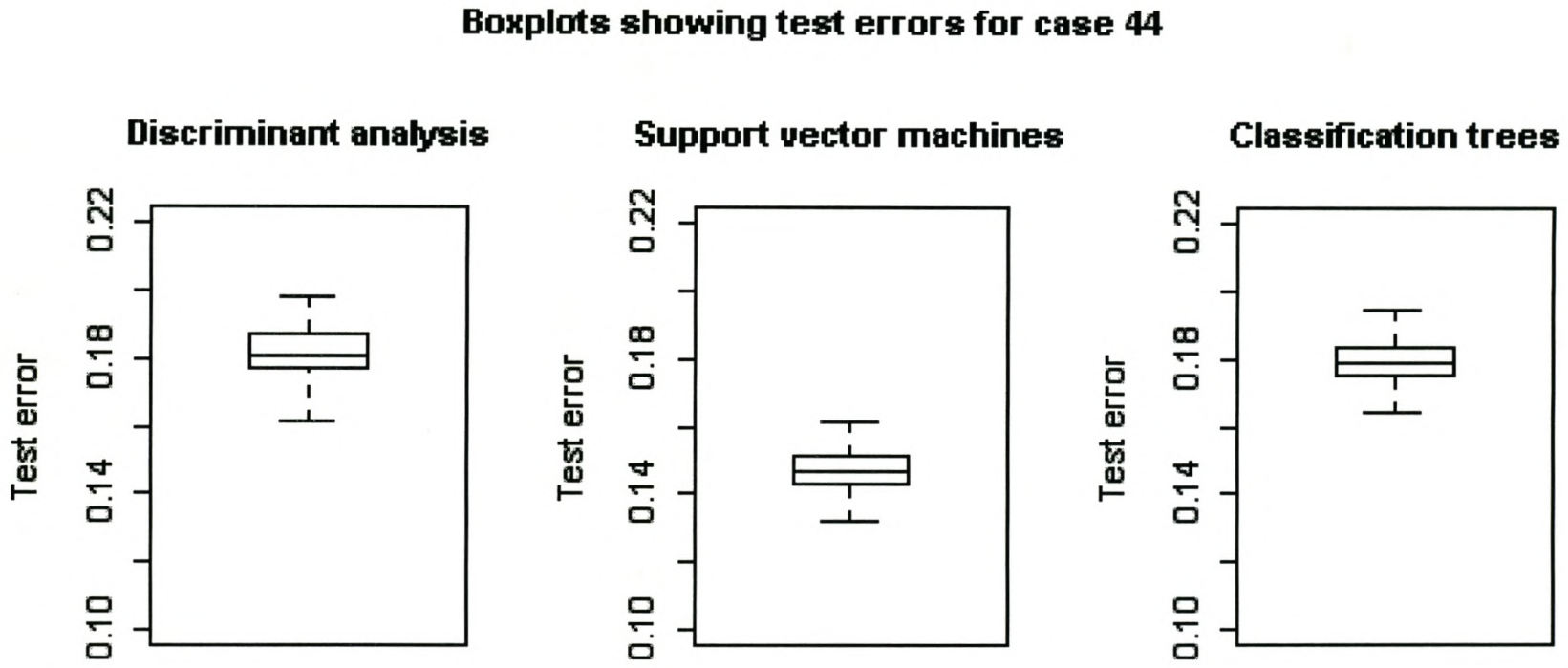
	LDA	SVM	Tree
Mean error rate	0.180933	0.146754	0.178296

CASE 45: Variable set A, RBF kernel, C = 10, γ = 0.0025



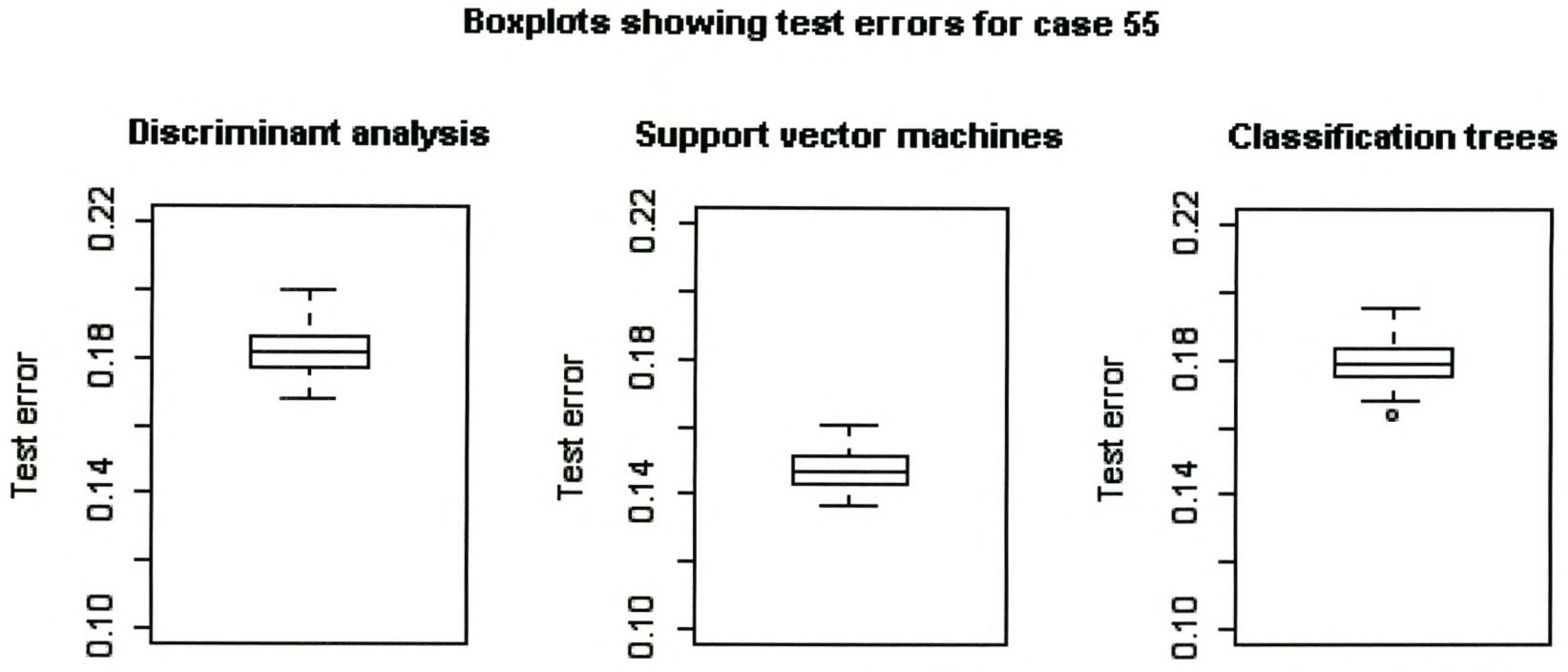
	LDA	SVM	Tree
Mean error rate	0.183592	0.146983	0.178888

CASE 18: Variable set A, RBF kernel, C = 1, $\gamma = 0.1$



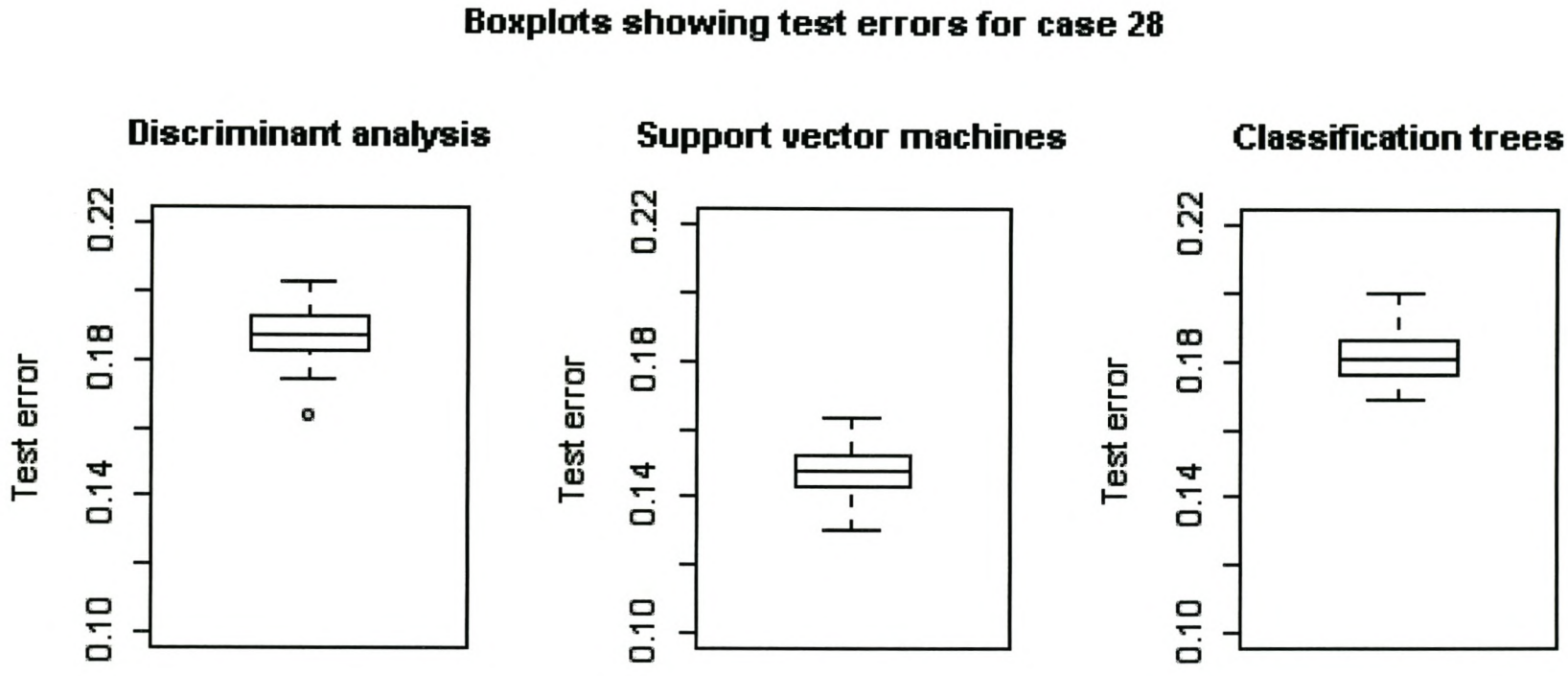
	LDA	SVM	Tree
Mean error rate	0.181825	0.147238	0.179621

CASE 44: Variable set A, RBF kernel, C = 40, $\gamma = 0.001$



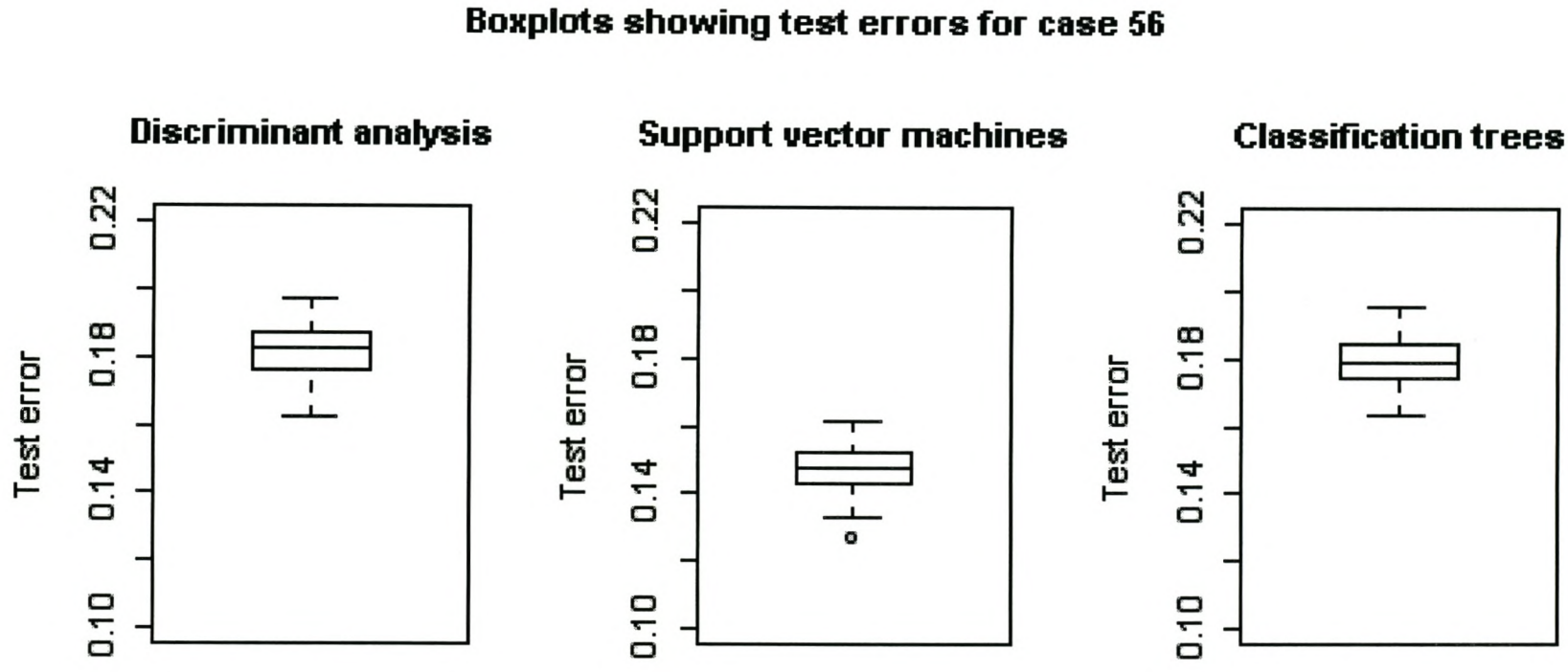
	LDA	SVM	Tree
Mean error rate	0.181742	0.147508	0.179379

CASE 55: Variable set A, RBF kernel, C = 30, γ = 0.0075



	LDA	SVM	Tree
Mean error rate	0.187571	0.147650	0.181371

CASE 28: Variable set B, RBF kernel, C = 100, $\gamma = 0.01$



	LDA	SVM	Tree
Mean error rate	0.181646	0.147663	0.179496

CASE 56: Variable set A, RBF kernel, C = 40, γ = 0.0075

The better performance of the SVM compared to the other two techniques shows that it is definitely a technique worth considering. Nevertheless, it is important to not only look at overall performance (i.e. the total misclassification rate), but also at the two types of misclassification, as one type of misclassification might have more serious consequences than the other. In our case, it is thought that the cost of not correctly identifying a possible lapse case (in other words, classifying a client as a non-lapse client when in fact the client is a potential lapse client) is more severe than classifying a non-lapse client as a potential lapse client.

The following three tables give the confusion matrices for linear discriminant analysis, support vector machines and classification trees. Variable set A was used, and for the SVM an RBF kernel with a cost parameter of 50 and a γ parameter of 0.001 was used.

LINEAR DISCRIMINANT ANALYSIS

		ACTUAL	
		Non-lapse	Lapse
PREDICTED	Non-lapse	41.12%	9.58%
	Lapse	8.49%	40.81%

Correct: 81.93%
Incorrect: 18.07%

SUPPORT VECTOR MACHINES

		ACTUAL	
		Non-lapse	Lapse
PREDICTED	Non-lapse	39.77%	4.78%
	Lapse	9.84%	45.61%

Correct: 85.38%
Incorrect: 14.62%

CLASSIFICATION TREES

		ACTUAL	
		Non-lapse	Lapse
PREDICTED	Non-lapse	37.39%	5.62%
	Lapse	12.22%	44.77%

Correct: 82.16%
Incorrect: 17.84%

In addition to being the technique that gave the lowest overall misclassification rate (compared to linear discriminant analysis and classification trees), SVMs were also the technique that gave the lowest error rate for incorrectly classifying a potential lapse client as a non-lapse client (only 4.78% of actual lapse clients were incorrectly predicted by the SVM).

From all of the above results it is clear that in this case support vector machines achieved better performance than some more traditional classification techniques. It is also clear that variable selection plays an important role in the performance of SVMs, and that the correct choice of parameters can also play a crucial role in the improvement of the SVM performance.

Chapter 6

Regression problem

1. Description of the basic problem

Income is an important variable in a client database. It can be used, amongst other things, to estimate the future profitability of a client, and to predict a client's propensity to buy a certain product. Household income is also one of the main variables by which Sanlam's target market is segmented. It is therefore important to have a reasonably accurate estimate of income on record for each client.

At a bank, a client usually has dealings with the company at least once a month. Contact between an insurance company and its clients is much more infrequent. Certain client personal details remain static over time. For instance, gender and race would not change; neither would date of birth or identification number. But a field such as income remains a problem, as it becomes outdated fairly quickly. In the case of a bank a good estimate of a client's income can usually be made from the monthly salary deposited into a client's bank account. In the case of insurance companies, this is not so simple. In most cases, the income on record for a client would be the income

obtained when the client last took out a policy – and this could be many years ago. The result is very outdated income information. Some clients also have no income on record, as it is not a compulsory information field that has to be completed for all products. Furthermore, the income carried on the database is personal income, whereas in most cases we are rather more interested in a client's household income.

Currently attempts have been made to update the client income records by means of the prevailing inflation rate every year, but this is not a very reliable method, as it completely ignores occurrences such as promotions and marriages (where household income would typically increase). For clients who have no income on record, an income figure is derived by looking at the size of the premiums the client pays. However, this is definitely not a very accurate measure.

Recently, all clients who took out a new product were sent a questionnaire. The purpose of the questionnaire was to get to know clients better. One of the questions on the questionnaire related to income. **The objective is thus to use the income data on the questionnaires combined with the personal details on record for these clients to attempt to predict household income for clients who did not take part in the survey.** This would mean that in future, a predicted income field could be added to the client records to try and overcome the problem of outdated or missing income data.

2. Definition of the input and response variables

The response variable in this study is the particular client's household income. However, in the questionnaire, exact incomes were not asked; instead, income was given in categories. Furthermore, there was an indicator for own (personal) income, spouse's income, as well as other income. Since we were interested in the household income of the client, an estimate for household income had to be derived first. This was done by taking the midpoint of each income category (for instance, if the income category was R2 000 – R4 000 monthly income, the midpoint would be R3 000) for each type of income (i.e. personal income, spouse's income and other income). These midpoints were then added together to obtain an estimate for total household income.

The input variables were obtained by matching the personal details on record for clients who completed a questionnaire with the derived income variable from the questionnaire. As in the case of the classification problem, data from two tables were used: personal details and aggregated policy information.

3. Data cleaning

Since the input variables were very similar to those used in the classification problem, the data cleaning process was very much the same. In other words, variables that were encoded in other variables were either deleted, or the aggregates were used. Categorical variables were also transformed to binary dummy variables. Because of smaller sample sizes, some additional variables were combined: for instance, in the

case of race the dummy variables for black, coloured and Asian were combined to compensate for small samples of these variables; the same was done with the dummy variables for title, marital status, language, province and channel.

4. Descriptive measures for the data

After data cleaning and variable transformation there were 47 variables (1 response and 46 input variables) and 3 652 data cases. Of the 46 input variables, 22 were categorical and 24 numerical. The response variable (income) was numerical.

On the following pages, some descriptive measures for the data are given. Firstly, a table with descriptions of each of the variables is given, followed by a table with descriptive statistics for the data (mean, median, standard deviation and range). The correlation of each input variable with the response variable is also given.

Variable name	Variable description	Variable type
TOT_INC	**Total household income (RESPONSE VARIABLE)**	**Numerical**
GEND_MALE	Gender indicator	Categorical
TITLE_ORD	Flag indicating an ordinary title (Mr. / Mrs. / Miss / Ms)	Categorical
TITLE_PROF	Flag indicating a professional title (e.g. doctor, advocate, etc.)	Categorical
TITLE_REV	Flag indicating a religious title (e.g. reverend)	Categorical
AGE	Age (in years)	Numerical
MRTL_ST_UNK	Flag indicating whether marital status is unknown	Categorical
MRTL_ST_SGL_NM	Flag indicating whether marital status is single (never been married)	Categorical
MRTL_ST_MAR	Flag indicating whether marital status is married / living together	Categorical
MRTL_ST_SGL_WM	Flag indicating whether marital status is divorced / widowed	Categorical
PERS_PPS_FLG	Flag indicating whether client is also a client of the Professional Provident Society	Categorical
LANG_AFR	Preferred language of client (1 = Afrikaans; 0 = English)	Categorical
RACE_WHITE	Race of client (1 = white; 0 = black / coloured / Asian)	Categorical
PROV_GAUTENG	Flag indicating whether client resides in Gauteng	Categorical
PROV_WC	Flag indicating whether client resides in the Western Cape	Categorical
PROV_KZN	Flag indicating whether client resides in Kwazulu Natal	Categorical
PROV_REST	Flag indicating whether client resides in Mpumalanga / Eastern Cape / Northern Cape / Free State / North West / Limpopo	Categorical
CHAN_ADV	Flag indicating whether client has an adviser (company representative) as intermediary	Categorical
CHAN_BROK	Flag indicating whether a client has a broker as intermediary	Categorical
CHAN_ORPH	Flag indicating whether client is orphaned (i.e. has no intermediary on record)	Categorical
CHAN_OTH	Flag indicating whether client has "other" intermediary (e.g. direct)	Categorical
ann_recur_prem	Total annual recurring premiums from client	Numerical
sngl_prems	Total single premiums from client	Numerical
monthly	Flag indicating whether client has any products that are paid monthly	Categorical
paid_up	Flag indicating whether client has any paid-up products	Numerical
time_W_S	Number of months has been a client of Sanlam	Numerical
time_R_S	Number of months client has remaining with Sanlam (longest remaining term on all policies)	Numerical
child_endow	Number of child endowment products	Numerical
endowm	Number of endowment products	Numerical
funeral	Number of funeral insurance products	Numerical
life_ann	Number of life annuity products	Numerical
OLV	Number of OLV products	Numerical
ra	Number of retirement annuities	Numerical
snkng_fnd	Number of sinking fund products	Numerical
trm_ann	Number of term annuity products	Numerical
w_life	Number of whole life products	Numerical
matrix	Number of Matrix products	Numerical
ut	Number of unit trust products	Numerical
T_LPOL	Time in months since client's last Sanlam product purchase	Numerical
lapsed	Flag indicating whether client has had a lapsed policy	Categorical
had_mat	Flag indicating whether client has had a matured policy	Categorical
tot_cov	Aggregate of sum of cover of all active products	Numerical
tot_cov2	Maximum of all active products' cover	Numerical
PLAN_CNT	Number of plans	Numerical
no_r_agmts	Number of recurring products	Numerical
no_s_agmts	Number of single premium products	Numerical
ave_time_between	Average time between Sanlam product purchases	Numerical

Variable name	Mean / mode[1]	Median[2]	SD[3]	Range		Corr.[4]
				Min	Max	
TOT_INC	**17039**	**16000**	**9853**	**1000**	**60000**	**1**
GEND_MALE	1		0.448	0	1	0.176
TITLE_ORD	1		0.214	0	1	-0.081
TITLE_PROF	0		0.169	0	1	0.095
TITLE_REV	0		0.109	0	1	-0.001
AGE	49	49	12.169	14	94	-0.138
MRTL_ST_UNK	0		0.233	0	1	0.026
MRTL_ST_SGL_NM	0		0.406	0	1	-0.171
MRTL_ST_MAR	1		0.476	0	1	0.224
MRTL_ST_SGL_WM	0		0.271	0	1	-0.159
PERS_PPS_FLG	0		0.198	0	1	0.109
LANG_AFR	1		0.466	0	1	-0.048
RACE_WHITE	1		0.295	0	1	0.089
PROV_GAUTENG	0		0.488	0	1	0.143
PROV_WC	0		0.421	0	1	-0.075
PROV_KZN	0		0.303	0	1	-0.024
PROV_REST	0		0.447	0	1	-0.070
CHAN_ADV	0		0.493	0	1	-0.080
CHAN_BROK	0		0.496	0	1	0.067
CHAN_ORPH	0		0.329	0	1	0.009
CHAN_OTH	0		0.146	0	1	0.022
ann_recur_prem	13101	9036	15322	0	212196	0.237
sngl_prems	37603	0	120042	0	2463793	-0.079
monthly	1		0.287	0	1	0.115
paid_up	0.194	0	0.581	0	6	0.033
time_W_S	202	177	116	7	748	0.016
time_R_S	5950	9999	4824	0	9999	-0.008
child_endow	0.166	0	0.532	0	5	0.051
endowm	1.421	1	1.360	0	9	0.002
funeral	0.030	0	0.173	0	2	-0.041
life_ann	0.101	0	0.362	0	4	-0.154
OLV	0.067	0	0.263	0	3	0.031
ra	1.261	1	1.211	0	11	0.180
snkng_fnd	0.057	0	0.316	0	6	-0.100
trm_ann	0.063	0	0.300	0	5	-0.122
w_life	0.563	0	0.935	0	10	0.094
matrix	0.097	0	0.351	0	4	-0.009
ut	0.424	0	0.920	0	12	-0.030
T_LPOL	49	29	56	-2	748	0.076
lapsed	0		0.238	0	1	-0.006
had_mat	0		0.462	0	1	-0.121
tot_cov	495200	260991	726349	0	8649000	0.224
tot_cov2	310727	163734	496172	0	7770000	0.199
PLAN_CNT	0.155	0	0.461	0	4	-0.186
no_r_agmts	3.297	3	2.355	0	19	0.176
no_s_agmts	0.494	0	0.995	0	9	-0.141
ave_time_between	39	31	33	0	367	-0.013

1. Mean for numerical variables; mode for categorical variables
2. For numerical variables only
3. Standard deviation
4. Correlation with response variable (TOT_INC)

5. Selection of variables

As already mentioned, there is currently no procedure in *R* for automatic selection of variables in a support vector machine. As with the classification problem, it was found that a model containing all 47 variables performed worse than a model with fewer variables, and therefore some form of variable selection needed to be done. The same approach to variable selection as in the classification case was followed. In other words, the data were read into SAS Enterprise Miner, and basic regression and basic tree models were built. Six variations of the models were built: 4 regression models and 2 trees. For the 4 regression models, 4 different variable selection methods were used: stepwise, backward, forward (all using the AIC criterion with a significance level of 0.05), as well as a backward selection method using validation error as criterion. The 2 tree models consisted of one allowing only binary splits, and one allowing a maximum of 5 branches per node.

The variables selected by each of these 6 methods were noted, and it was decided to use only those variables selected by at least 3 of the 6 models.

The 20 variables (over and above the dependent variable) that were selected are: GEND_MALE, AGE, MRTL_ST_MAR, MRTL_ST_SGL_WM, PERS_PPS_FLG, LANG_AFR, RACE_WHITE, PROV_GAUTENG, ann_recur_prem, sngl_prems, ra, w_life, T_LPOL, tot_cov, PLAN_CNT, TITLE_PROF, MRTL_ST_SGL_NM, CHAN_BROK, paid_up and ave_time_between.

6. Model and parameter selection

An R program was written (see Appendix B) to perform the data analysis. Preliminary analysis using a neural network in Enterprise Miner showed that this technique did not perform as well as a predictor of income as multiple linear regression and regression trees did; therefore, only multiple linear regression, support vector machines and trees were implemented for the comparison in R.

As in the classification case, data were randomly divided into a training set and a test set. A model was fit to the training data, and then the model was used to predict income values for the cases in the test data set. The prediction error was calculated as the sum of the squared differences between the actual income values and the predicted income values. For each model, 100 iterations were performed, and the average error calculated over the 100 iterations.

No parameter tuning was performed for the multiple linear regression and tree models; for SVMs, several combinations of parameters were tried.

The following parameters were varied:

- ε – regression vs. ν – regression (with values of 0.2, 0.5 and 0.8 for ε and ν) (see Section 6.3, Chapter 3 for a discussion of the differences between ε – regression and ν – regression)
- no transformation of the response variable vs. a log transformation of the response variable
- cost parameter (C) of the SVM (values of 1, 40 and 100)

- gamma parameter (γ) of the SVM (values of 0.01, 0.001 and 0.00001)

As it was clear from the classification results that the radial basis function kernel performed best, and also because Meyer (2002) states that the RBF kernel is currently the only one that is optimised for in R, other kernels were not implemented.

7. Results

The results for the multiple linear regression model and regression trees were reasonably constant. For the 18 cases ran without the log transformation of the response variable, the descriptive statistics for the mean errors obtained by these two techniques were as follows:

Table 6.1: Descriptive statistics for mean error rates for multiple linear regression and regression trees

Technique	Mean	Standard deviation	Range	
			Minimum	Maximum
Linear model	76 942 180	397 741	76 398 113	77 875 032
Tree	86 289 840	444 237	85 375 231	86 965 435

With a log transformation of the response variable, both the linear model and the regression tree performed more or less the same (mean errors of 76 877 794 and 86 260 878 respectively).

The very small variation in the errors for the regression trees and the multiple linear regression model suggested that the errors for the SVMs using different parameters

could be compared directly to find the best performing parameter set, since the sample variation appeared to be very small.

The errors for these parameter combinations are given in the tables below (errors are rounded to the nearest 1000, and in each case ε and ν equal to 0.5 was used). It is clear that, as in the classification problem, there is no clear-cut "best" choice for C and γ, but rather that certain combinations of these parameters perform well.

Tables 6.2 – 6.5: Errors for different values of cost and gamma, different variable sets and ε – regression vs. ν – regression

ε–regression; no transformation of response variable

C	0.001	γ = 0.00001	0.01
40	75 989	77 210	79 842
1	77 897	84 544	75 645
100	75 945	77 231	83 651

ε–regression; log transformation of response variable

C	0.001	γ = 0.00001	0.01
40	78 739	81 779	83 741
1	80 158	99 455	78 579
100	78 862	82 812	91 343

ν–regression; no transformation of response variable

C	0.001	γ = 0.00001	0.01
40	76 759	78 957	79 204
1	77 891	95 435	75 982
100	75 423	77 881	83 219

ν–regression; log transformation of response variable

C	γ = 0.001	0.00001	0.01
40	79 746	83 894	85 018
1	82 046	99 005	77 954
100	78 226	81 723	92 738

It was also clear that the log transformation of the response variable caused the SVM to perform worse. Furthermore, there was no substantial difference between ε and ν regression at the 0.5 level. Further cases were attempted with ε and ν values of 0.2 and 0.8, but they proved to perform worse than the 0.5 cases.

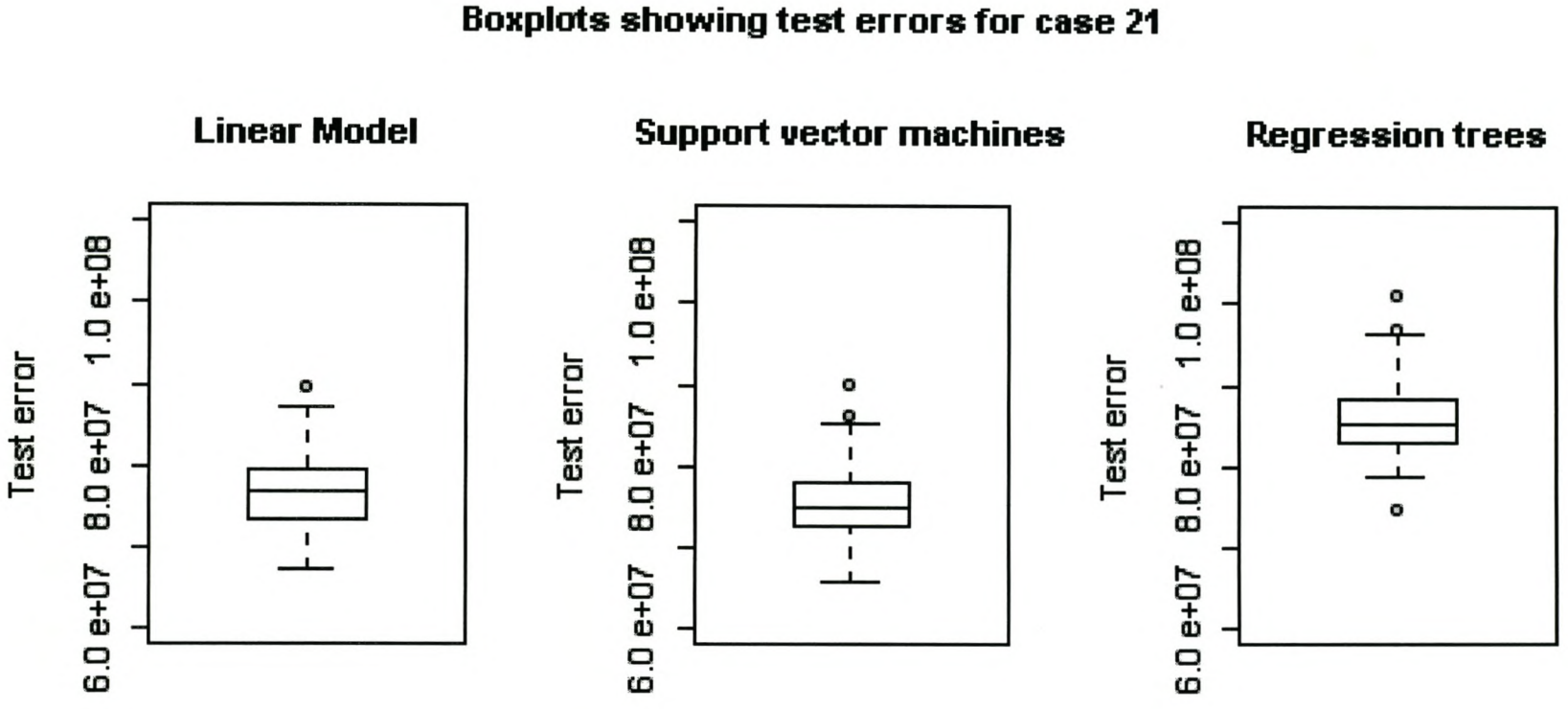
The best performing SVM case overall was the one with a cost parameter of 100 together with a gamma value of 0.001, using ν-regression with no log transformation of the response variable. This was also lower than the lowest error rate obtained by the other two techniques. However, other than was the case with the classification problem (where the SVM proved to outperform the other techniques, no matter what parameter choice), the SVM only outperformed the multiple linear regression model in 6 instances. (In contrast, it only did worse than the regression tree in 5 instances). The parameter choices for the 6 cases where the SVM outperformed the multiple linear regression model are given in the table below:

Table 6.6: Parameter values, regression type and variable set for best performing SVM cases

Case number	C	γ	Regression type	Transformation of response?
21	100	0.001	ν	No
8	1	0.01	ε	No
3	100	0.001	ε	No
26	1	0.01	ν	No
1	40	0.001	ε	No
19	40	0.001	ν	No

On the following pages, box plots of the error rates for the 5 best and the 5 worst performing SVM cases are given, together with the corresponding box plots for the multiple linear regression model and regression trees. It is apparent that parameter choice for the SVM is very important, and an incorrect choice of parameters can have a dramatically adverse effect on SVM performance.

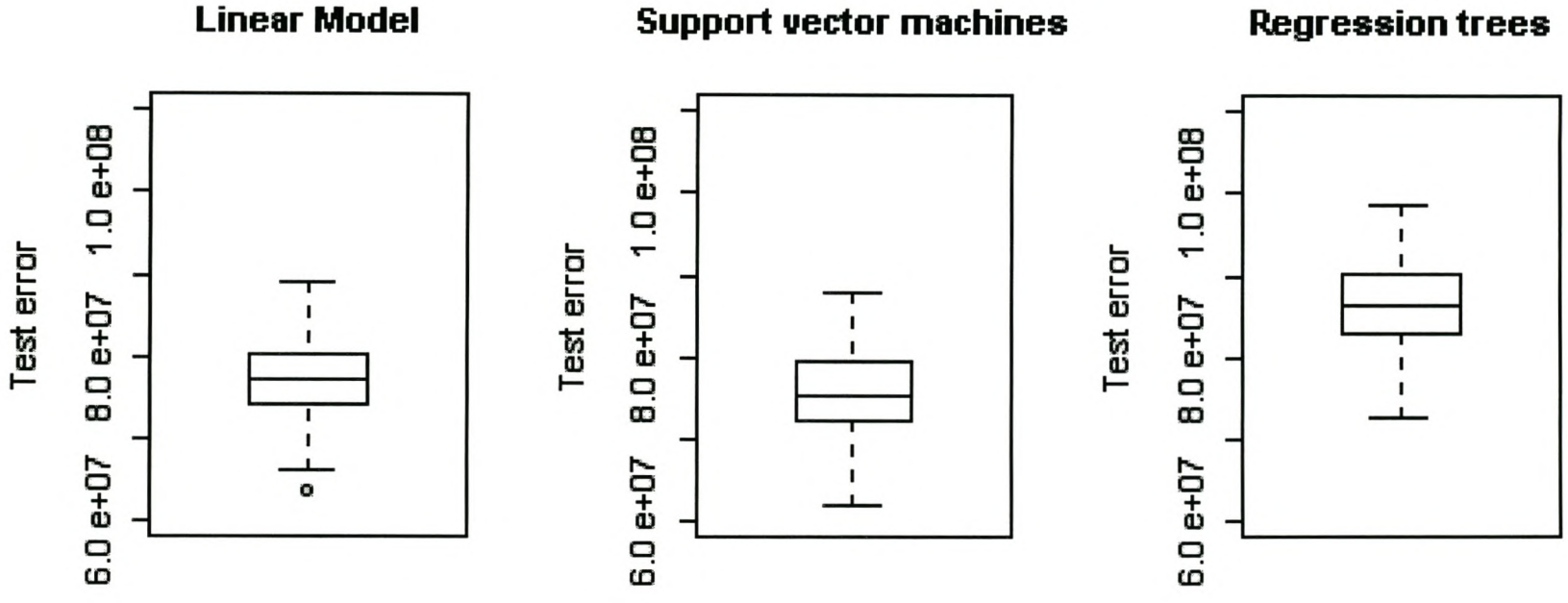
Still, the better performance of the SVM compared to the other two techniques in some of the cases confirms that it is definitely a technique worth considering.



	Linear model	SVM	Tree
Mean squared error	76 754 775	75 422 910	85 849 131

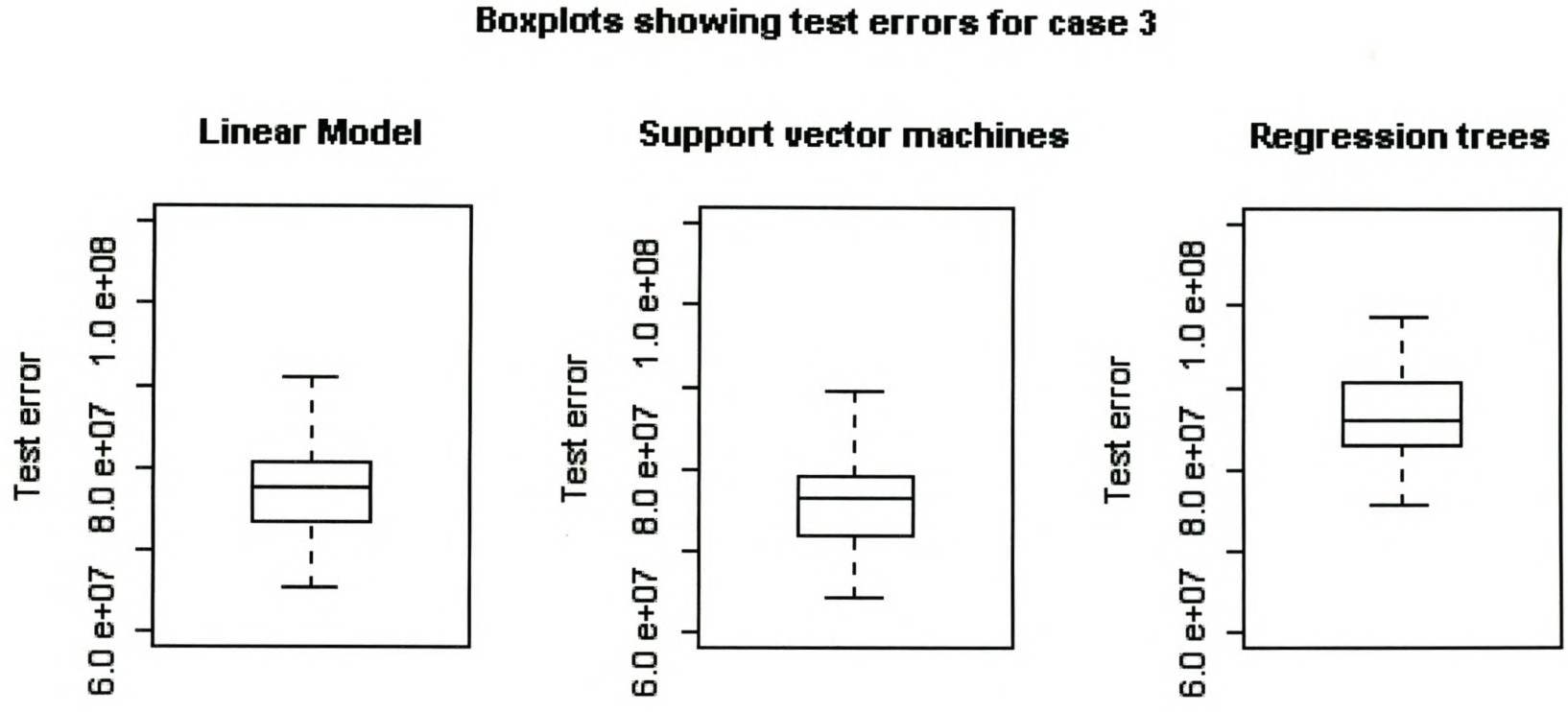
CASE 21: No transformation of response, ν regression, C = 100, γ = 0.001

Boxplots showing test errors for case 8



	Linear model	SVM	Tree
Mean squared error	77 113 900	75 644 728	86 325 993

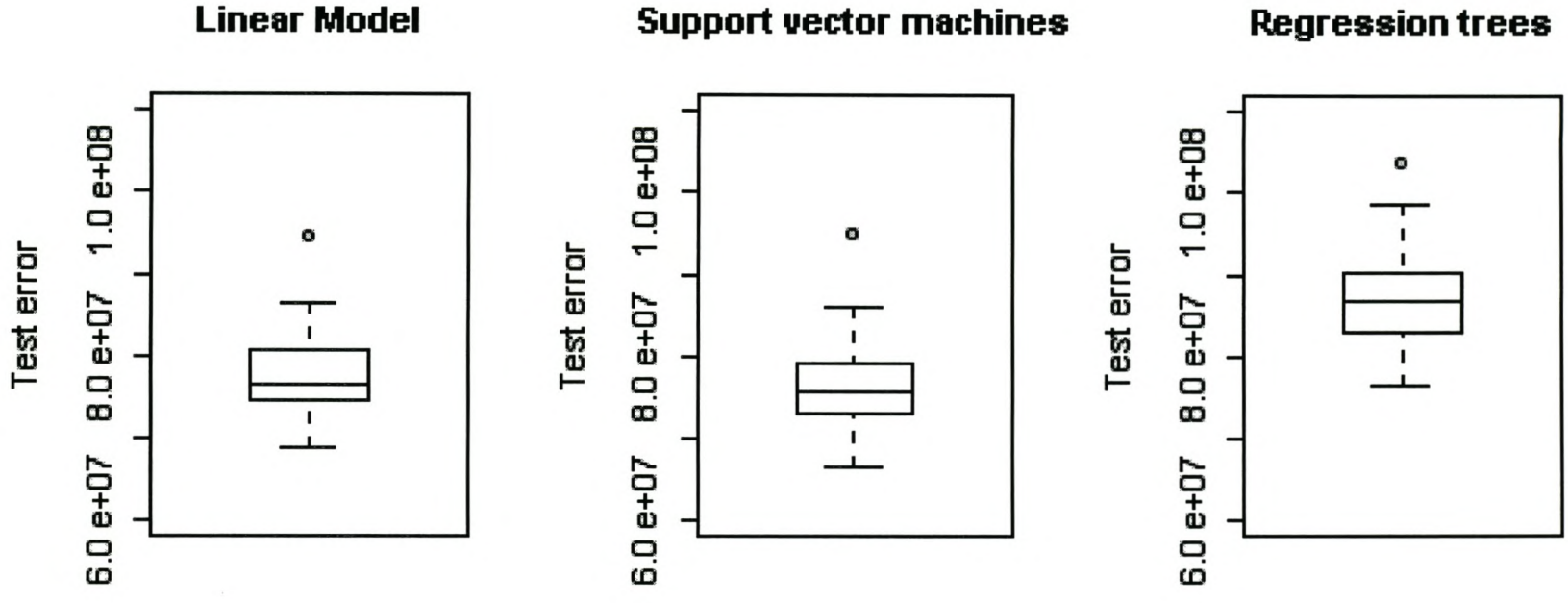
CASE 8: No transformation of response, ε regression, C = 1, γ = 0.01



	Linear model	SVM	Tree
Mean squared error	77 342 204	75 944 648	86 610 441

CASE 3: No transformation of response, ε regression, C = 100, γ = 0.001

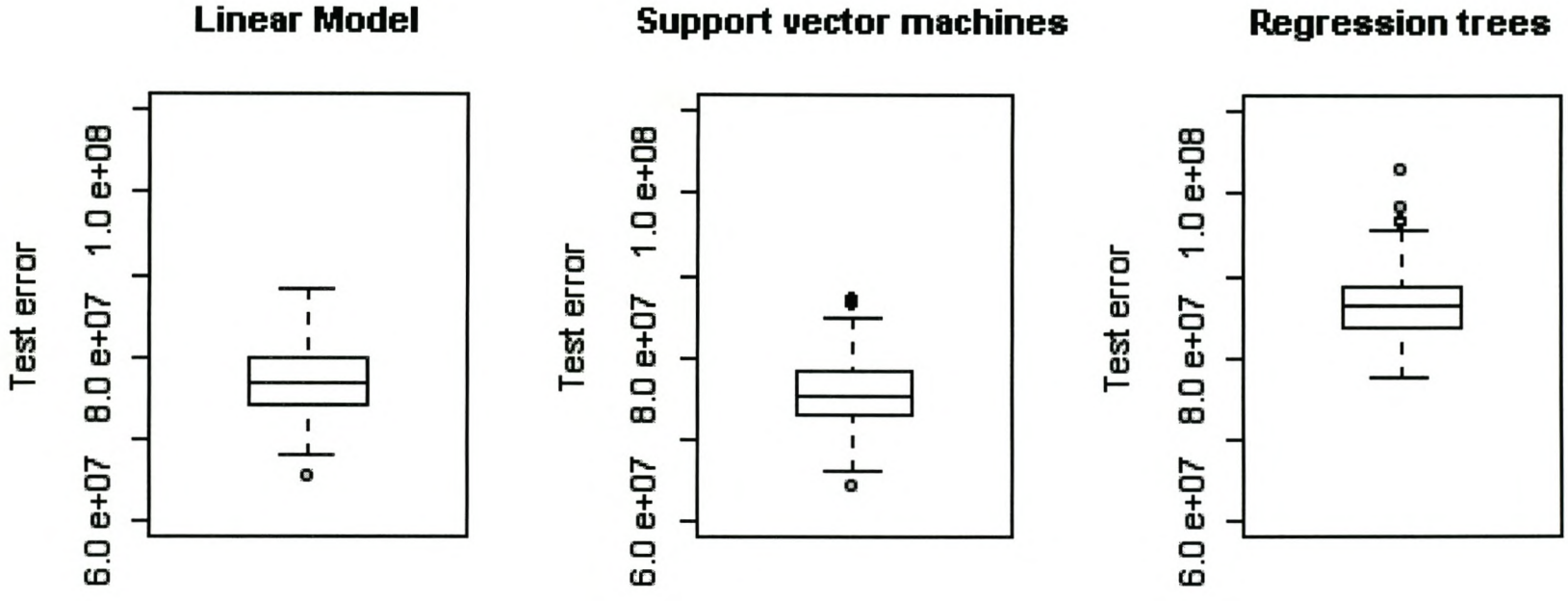
Boxplots showing test errors for case 26



	Linear model	SVM	Tree
Mean squared error	77 296 400	75 982 253	86 798 800

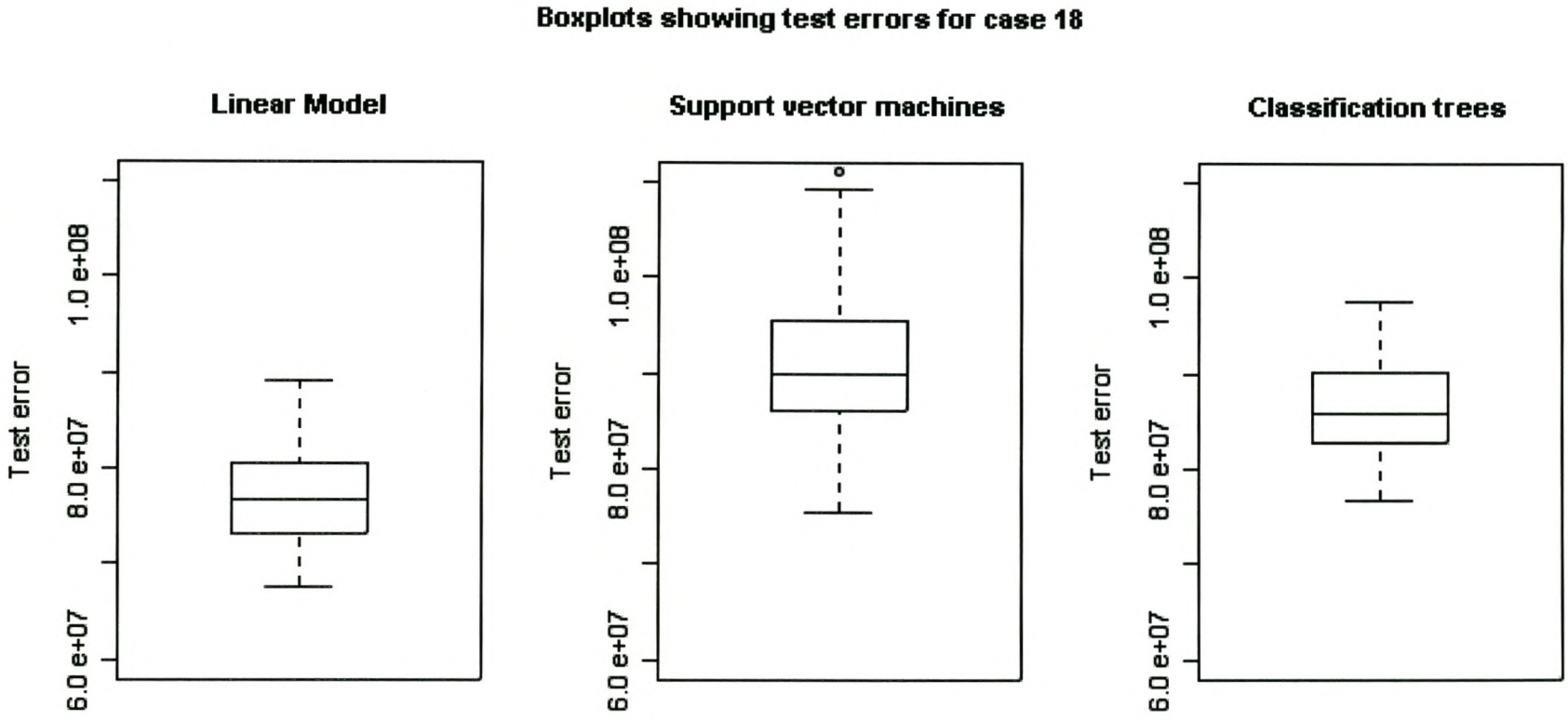
CASE 26: No transformation of response, ν regression, C = 1, γ = 0.01

Boxplots showing test errors for case 1



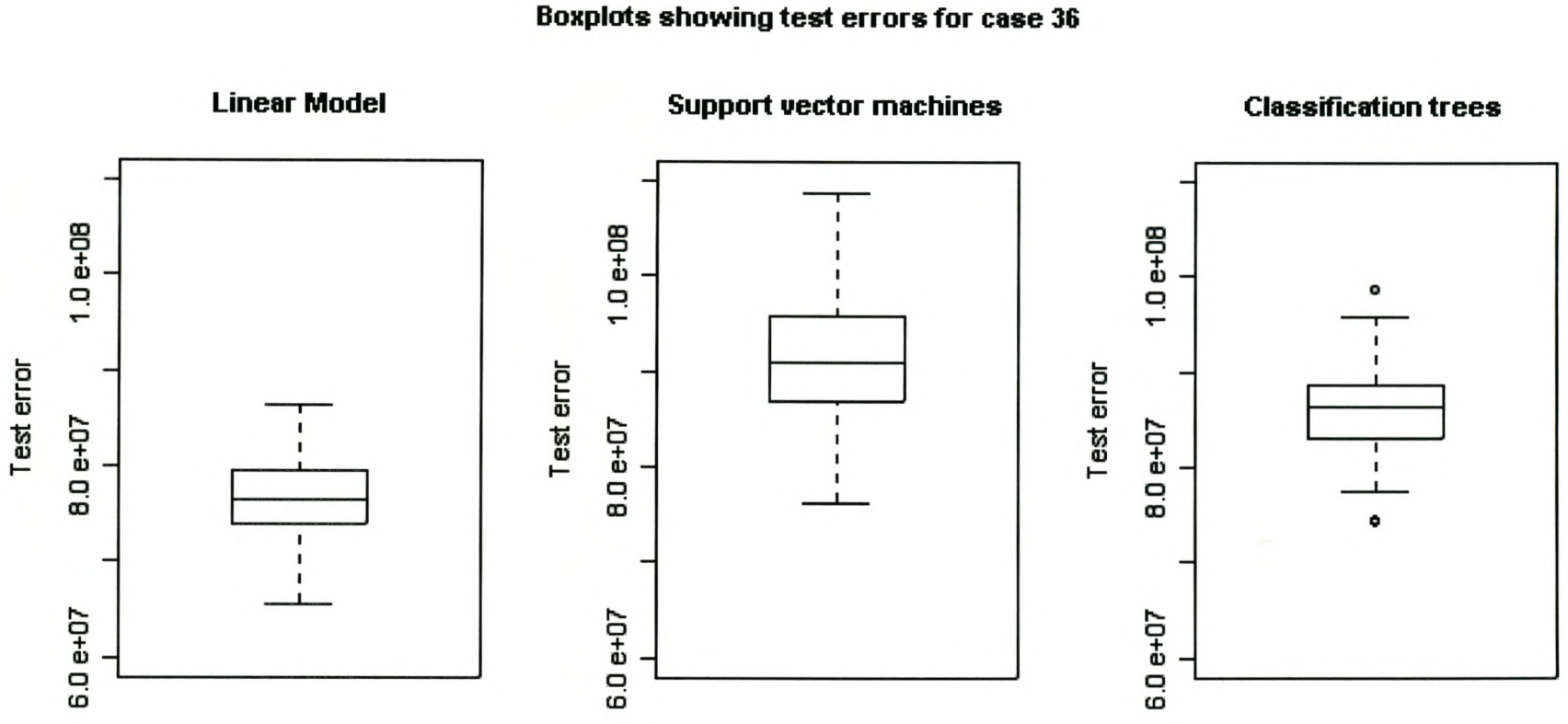
	Linear model	SVM	Tree
Mean squared error	77 160 905	75 989 129	86 599 773

CASE 1: No transformation of response, ε regression, C = 40, γ = 0.001



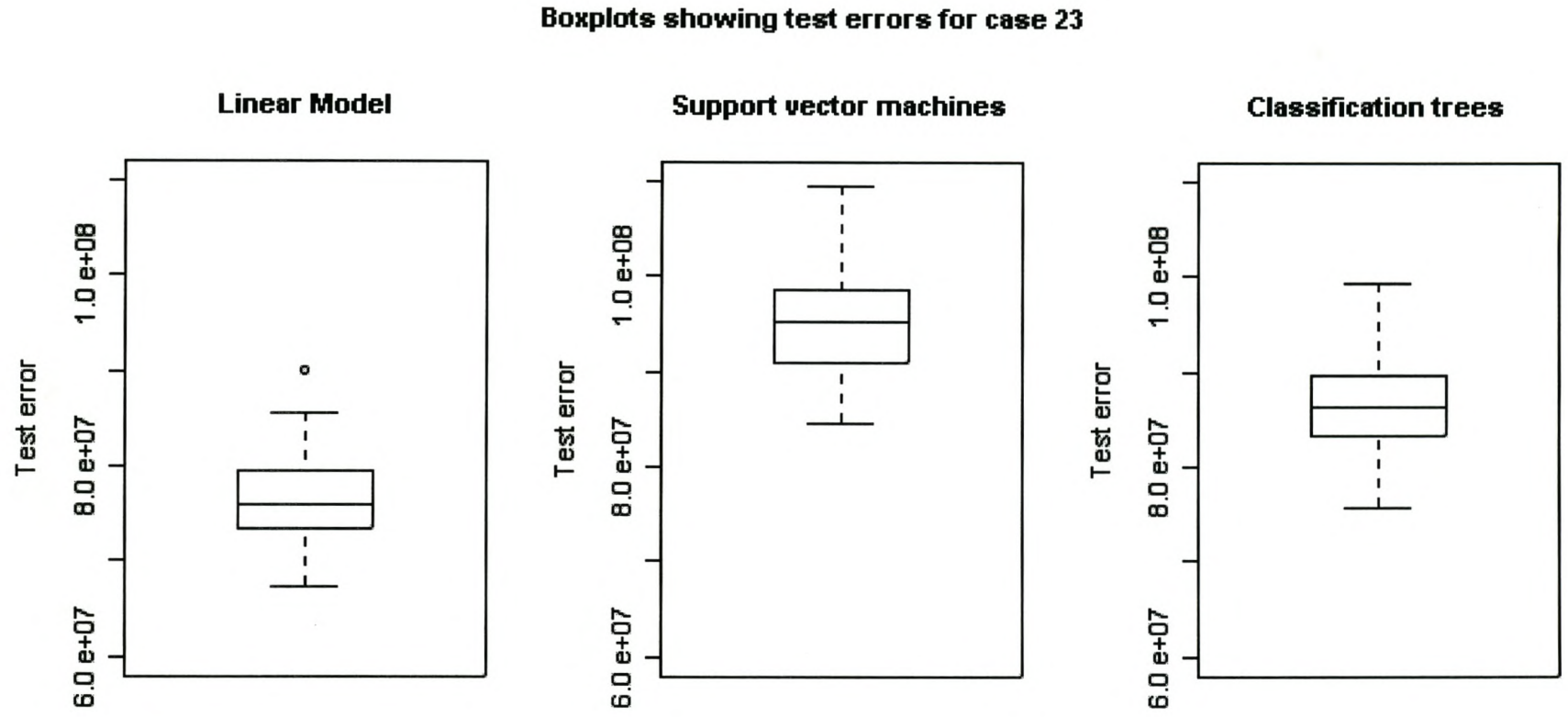
	Linear model	SVM	Tree
Mean squared error	76 936 745	91 343 166	86 404 003

CASE 18: Log transformation of response, ε regression, C = 100, γ = 0.01



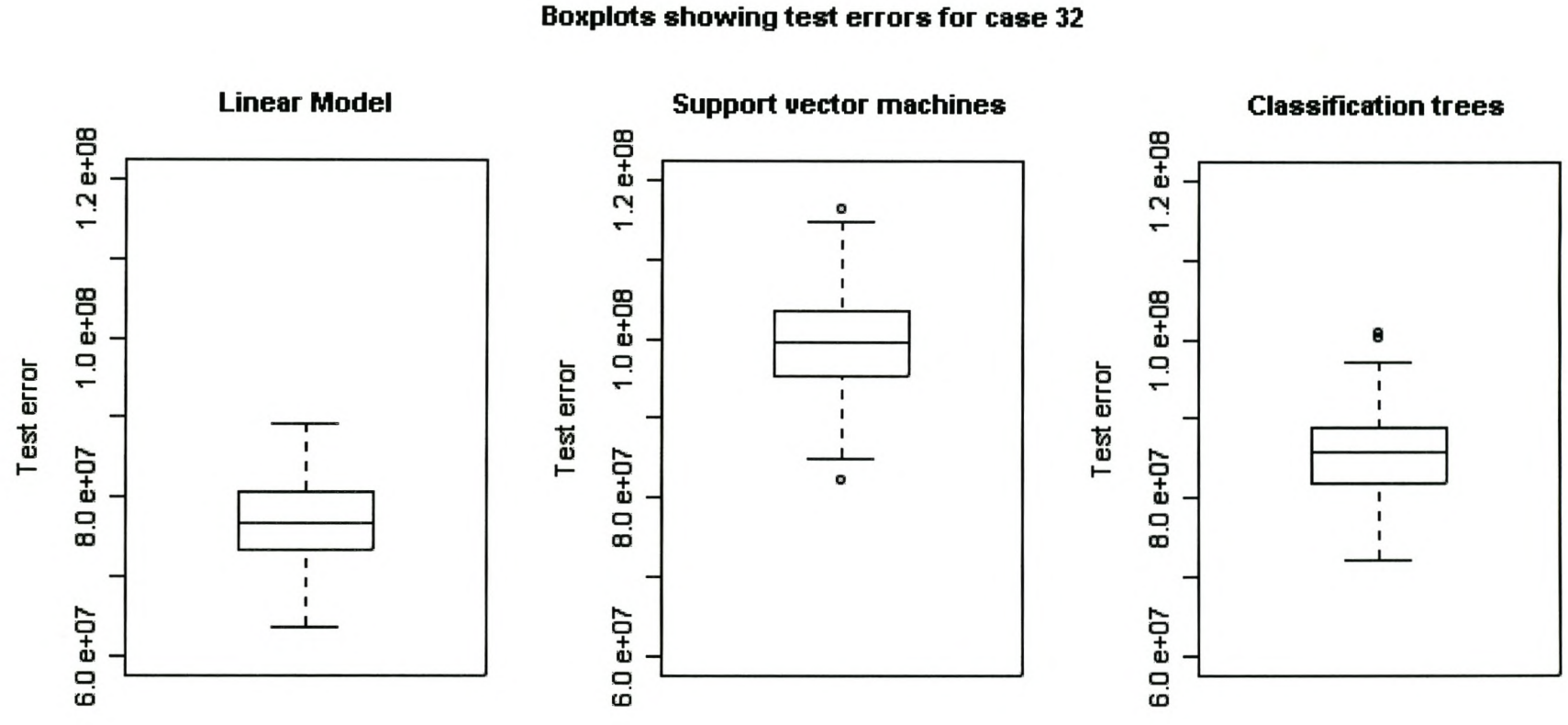
	Linear model	SVM	Tree
Mean squared error	76 539 108	92 737 582	86 179 514

CASE 36: Log transformation of response, ν regression, C = 100, γ = 0.01



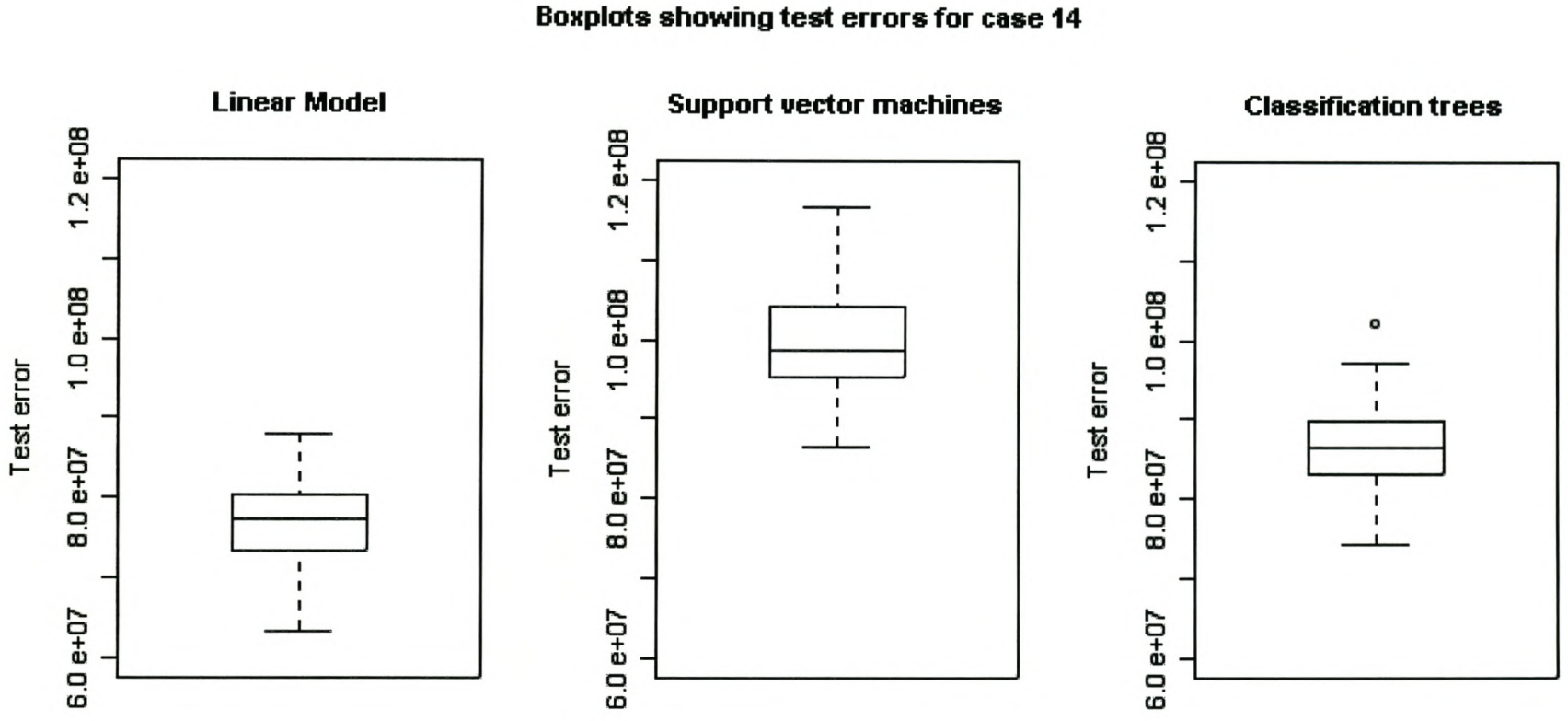
	Linear model	SVM	Tree
Mean squared error	76 560 205	95 435 147	86 322 916

CASE 23: No transformation of response, ν regression, C = 1, γ = 0.00001



	Linear model	SVM	Tree
Mean squared error	76 661 515	99 005 211	85 610 901

CASE 32: Log transformation of response, ν regression, C = 1, γ = 0.00001



	Linear model	SVM	Tree
Mean squared error	77 092 418	99 455 036	86 578 101

CASE 14: Log transformation of response, ε regression, C = 1, γ = 0.00001

In spite of the better performance of the SVM in some of the cases considered, overall errors were still very high. This is probably due to a variety of reasons, among which the large variation in the response variable, small sample sizes and insufficient data. Income is unquestionably a difficult variable to estimate, and the large errors obtained during this study show that the estimation is not really meaningful in this instance. However, if it is not possible to predict the actual income level with a reasonable level of precision, it may be sufficient to correctly estimate the income *segment* of a client.

At Sanlam, four income segments are usually employed: entry-level individuals, low middle individuals, high middle individuals and the affluent. The actual and predicted income values were examined to determine whether *the income segment* of a client was at least correctly estimated in most instances.

The following three tables show the confusion matrices for the multiple linear regression models, support vector machines and regression trees for the best performing SVM case (that is, case 21, which is a ν-regression model with no transformation of the response variable, a cost parameter of 100 and a γ parameter of 0.001).

LINEAR MODEL

		ACTUAL			
		Entry-level	Low middle	High middle	Affluent
PREDICTED	Entry-level	2.30%	0.55%	0.11%	0.00%
	Low middle	10.84%	10.62%	6.35%	0.66%
	High middle	5.81%	17.52%	35.71%	8.43%
	Affluent	0.00%	0.11%	0.77%	0.22%

Correct: 48.85%

SUPPORT VECTOR MACHINE

		ACTUAL			
		Entry-level	Low middle	High middle	Affluent
PREDICTED	Entry-level	3.29%	0.88%	0.33%	0.00%
	Low middle	11.06%	13.03%	8.76%	0.88%
	High middle	4.60%	14.79%	33.52%	8.43%
	Affluent	0.00%	0.11%	0.33%	0.00%

Correct: 49.84%

REGRESSION TREE

		ACTUAL			
		Entry-level	Low middle	High middle	Affluent
PREDICTED	Entry-level	0.00%	0.00%	0.00%	0.00%
	Low middle	12.81%	15.01%	12.38%	1.75%
	High middle	6.13%	13.80%	30.56%	7.56%
	Affluent	0.00%	0.00%	0.00%	0.00%

Correct: 45.56%

The above tables clearly show that there is a large extent of misclassification for all three of the techniques – in all cases, less than 50% of clients are classified into the correct income segment. The tree fares especially poorly, and it classified all clients into either the low middle or high middle segments.

For the multiple linear regression model and support vector machines, it was then decided to combine the low middle and high middle segments into a total middle segment. Classification performance is then improved, as the following two tables show. However, the improvement in classification performance is probably due to the fact that the majority (about 75%) of the sample cases actually were in the middle segment.

LINEAR MODEL

		ACTUAL		
		Entry-level	Middle	Affluent
PREDICTED	Entry-level	2.30%	0.66%	0.00%
	Middle	16.65%	70.21%	9.09%
	Affluent	0.00%	0.88%	0.22%

Correct: 72.73%

SUPPORT VECTOR MACHINE

		ACTUAL		
		Entry-level	Middle	Affluent
PREDICTED	Entry-level	3.29%	1.20%	0.00%
	Middle	15.66%	70.10%	9.31%
	Affluent	0.00%	0.44%	0.00%

Correct: 73.39%

The problem of predicting a client's household income or household income segment has therefore not been satisfactorily resolved. However, only a relatively small sample was available for this study, and it would probably be worthwhile to revisit the problem once more data (from client questionnaires) have become available. Another approach could be to approach the problem from the outset as a classification rather than a regression problem.

Chapter 7

Conclusions and further work

1. Issues that warrant further attention

It is clear from the empirical study reported in this thesis that support vector machines outperformed more traditional statistical techniques in the two problems that were considered. The supremacy of the SVM was especially evident in the classification problem. This is in line with findings reported by many other authors. Schölkopf and Smola (2002, p.22) refer to several studies reported in the literature where SVMs were found to be competitive with the best available procedures in problems such as optical character recognition, time series prediction and inverse function estimation. Another recent reference in this regard is Stecking and Schebesch (2003), who used SVMs in a credit scoring context, and found it to provide more accurate results than linear discriminant analysis and logistic regression.

In view of the relative newness of the SVM technique, there are still many issues that require further investigation. An important issue that has hitherto been largely neglected is that of input variable selection. Hastie et al. (2001, pp. 384-385) provide

evidence in support of the claim that the classification accuracy of SVMs is detrimentally affected by the presence of noise variables. This is in line with findings from our empirical study: using only a subset of "appropriate" input variables instead of all those that were available, leads to lower test errors. The crucial question in this regard concerns the procedure(s) that should be used in a given problem to identify a subset of "appropriate" input variables. Some work has already been done on this issue; see for example Grandvalet and Canu (2002), and Sugiyama and Müller (2002). Without going into detail, it seems that there are two basic approaches. *Firstly*, input variable selection can be based on an initial variable screening process where the fact that the data will subsequently be analysed using a support vector machine is not taken into account. This approach was used in our empirical study when we performed variable selection based, for example, on the correlation between the response and the input variables. A *second* approach is to integrate input variable selection into the process of fitting a support vector machine, thereby acknowledging that the process of variable selection should take cognisance of the fact that selection will be followed by analysis using SVMs. The latter approach is most probably preferable. On the whole however, the issue of input variable selection in SVMs has not been resolved, and further investigation is necessary. Note, finally, in this regard the distinction between *input* variable selection and *feature* selection. The former refers to the original untransformed independent or predictor variables, while the latter refers to selection after the input variables have been transformed to feature space using a kernel function.

Hyperparameter specification is also an important issue for SVMs. By hyperparameters we mean quantities such as the kernel function parameters, the cost

parameter and the ε parameter in the regression case. Many proposals have been made in this regard; see for example Gunn (1998), Schölkopf and Smola (2002, pp. 216-218), and Hsu et al. (2003). Often, the "best" way suggested for parameter specification is cross-validation. This can, however, be very expensive in terms of training time, and other ways of specifying SVM parameters are currently being sought.

2. Related new techniques

Support vector machines owe their successful implementation in classification and regression problems in no small measure to the kernel method for computing inner products in feature spaces. It is by using a kernel function that we are able to fit a hyperplane in feature space, thereby effectively fitting a nonlinear decision boundary or a nonlinear regression function in input space. The use of the kernel trick is however not limited to SVMs. In fact, this technique can be used to develop nonlinear generalisations of any linear algorithm that can be formulated in such a way that the data appear in the algorithm only in terms of inner products. It is therefore not surprising that several non-SVM algorithms based on application of the kernel trick have been developed and successfully implemented. Important references in this regard are Part 3 of Schölkopf and Smola (2002), and Herbrich (2001). Without going into any detail, we now briefly refer to some of these so-called *kernel methods*.

Kernel principal component analysis (kernel PCA) is described in Chapter 14 of Schölkopf and Smola (2002). Its aim is to extract principal components in feature

space, i.e. principal components of the variables that are nonlinearly related through some feature mapping to the original input variables. This is accomplished as in ordinary principal component analysis by solving an eigenvalue problem. Schölkopf and Smola (2002) describe the algorithm, indicating, inter alia, how to centre data in feature space (a requirement of kernel PCA), and discussing applications of kernel PCA.

Kernel Fisher discriminant analysis (KFDA) is discussed in Chapter 15 of Schölkopf and Smola (2002), and in Herbrich (2001, pp. 103-109). As can be deduced from the name of this technique, it represents a so-called kernelised version of the well-known linear discriminant analysis procedure introduced by Fisher (1936). By using the kernel trick, KFDA in principle constructs a linear function of the features that maximally separates the two groups while simultaneously minimising the within groups feature space variation. Indications are that the classification performance of KFDA (in terms of its ability to correctly classify new cases into the two groups) is comparable to that of SVMs, and superior to that of ordinary discriminant analysis in cases where a linear decision boundary does not separate the two groups well.

The *relevance vector machine* was introduced by Tipping (2000, 2001), and is discussed in Schölkopf and Smola (2002, pp.506-511) and in Herbrich (2001, pp. 92-97). This procedure is an example of a Bayesian approach to kernel methods. It can be applied in regression and in classification, and is based on the prior assumption that the coefficients α_i in a support vector machine are independent normal random variables centred at zero. Various approaches for obtaining the required approximate posterior quantities are discussed in the literature.

There are several other kernel-based procedures that have been introduced in the literature and that are being applied to solve practical problems. These include *kernel logistic regression* and a variant thereof, the *import vector machine* (see Zhu and Hastie, 2001), and the *Bayes point machine* together with a Markov Chain Monte Carlo algorithm for determining the required posterior quantities, the *kernel billiard algorithm* (see Herbrich, 2001, pp.97-103). Although these exotic sounding procedures generally perform well, they are sometimes based on fairly ad hoc arguments, and there seems to be scope for development of more formal statistical theory in this regard.

3. Concluding remarks

In most business environments, even a very small improvement in model performance can have significant financial implications. New techniques for both classification and regression are continuously being developed by statisticians and experts in applied areas such as machine learning. However, there is usually a significant lag between the introduction of these techniques in the literature and their implementation in commercially available software packages. One such example is the support vector machine, which was introduced in the early nineties, but took about 10 years to be implemented in commercial software packages. It is therefore an important – but also difficult – challenge for companies to remain up to date with the newest techniques that become available, and to keep abreast of new developments. In this context the following pronouncement seems particularly appropriate:

"In the 21st century, corporate survival will depend on how well vast amounts of data are mined."

- Dr. Jim Goodnight, President and Co-Founder SAS Institute Inc.

Appendix A

Mathematical background

In this appendix we briefly review some mathematical results that play an important role in the study of support vector machines. These pertain mainly to hyperplanes and convex optimisation. The discussion is brief, and the interested reader is referred to Rockafellar (1970) and Kroon (2003) for more comprehensive treatments of the topics.

1. Some concepts from convex analysis

Although convex analysis can be developed in abstract vector spaces, our attention will be restricted to the well known vector space R^n consisting of all $n-$tuples or real vectors $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, where $x_i \in R$, $i = 1, 2, \ldots, n$. The inner (or dot) product between two vectors $\mathbf{x}$ and $\mathbf{y}$ belonging to this vector space will be denoted by $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}'\mathbf{y}$, i.e. $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}'\mathbf{y} = \sum_{i=1}^{n} x_i y_i$. The *line* through two different vectors

$\mathbf{x}$ and $\mathbf{y}$ in R^n is defined to be the set of all vectors of the form $\lambda\mathbf{x}+(1-\lambda)\mathbf{y}$, $\lambda \in R$. A subset M of R^n is called *affine* if $\lambda\mathbf{x}+(1-\lambda)\mathbf{y} \in M$ for all $\mathbf{x}$ and $\mathbf{y}$ in M and all $\lambda \in R$. It can now be shown that the *subspaces* of the vector space R^n are the affine sets in R^n that contain the origin (see Rockafellar, 1970, Theorem 1.1).

The concept of a *hyperplane* is important in the development of support vector machines. This concept can be defined as follows in terms of an affine set. Let $M \subset R^n$ and $\mathbf{a} \in R^n$. Then the *translate* of M by $\mathbf{a}$ is the set $M+\mathbf{a} = \{\mathbf{x}+\mathbf{a} : \mathbf{x} \in M\}$. An affine set M is now said to be *parallel* to an affine set L if $M = L+\mathbf{a}$ for some vector $\mathbf{a}$. Rockafellar (1970, Theorem 1.2) shows that each non-empty affine set M is parallel to a unique subspace L. The *dimension* of a non-empty affine set is defined to be the dimension of the subspace that is parallel to it. A *hyperplane* is defined as an $(n-1)-$dimensional affine set in R^n.

The following argument leads to a convenient characterisation of a hyperplane. The $(n-1)-$dimensional subspaces of R^n are the orthogonal complements of the one-dimensional subspaces. A basis of a one-dimensional subspace consists of a single non-zero vector $\mathbf{w}$, and the $(n-1)-$dimensional subspaces of R^n are therefore sets of the form $\{\mathbf{x} : \mathbf{x} \perp \mathbf{w}\}$, where the notation $\mathbf{x} \perp \mathbf{w}$ indicates that the vectors $\mathbf{x}$ and $\mathbf{w}$ are orthogonal. We now have

$$\begin{aligned}\{\mathbf{x} : \mathbf{x} \perp \mathbf{w}\} + \mathbf{a} &= \{\mathbf{x} + \mathbf{a} : \langle \mathbf{x}, \mathbf{w} \rangle = 0\} \\ &= \{\mathbf{x} : \langle \mathbf{x} - \mathbf{a}, \mathbf{w} \rangle = 0\} \\ &= \{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle = b\}\end{aligned}$$

where $b = \langle \mathbf{a}, \mathbf{w} \rangle$. Any hyperplane can therefore be represented in the form $\{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle = b\}$, where the vector $\mathbf{w}$ and the scalar b are unique up to multiplication by a non-zero constant. The vector $\mathbf{w}$ is called a *normal* to the hyperplane. Finally, the signed distance from an arbitrary point $\mathbf{z} \in R^n$ to the hyperplane $\{\mathbf{x} : \langle \mathbf{x}, \mathbf{w} \rangle + b = 0\}$ is given by $\dfrac{\langle \mathbf{z}, \mathbf{w} \rangle + b}{\|\mathbf{w}\|}$, where $\|\mathbf{w}\|^2 = \sum_{i=1}^{n} w_i^2$ is the squared norm of $\mathbf{w}$.

We now turn to some definitions and results regarding *convex* sets and functions. A set A in R^n is said to be *convex* if $\lambda \mathbf{x} + (1-\lambda)\mathbf{y} \in A$ for all $\mathbf{x}$ and $\mathbf{y}$ in A and all $0 < \lambda < 1$. It can be shown that the intersection of an arbitrary collection of convex sets is once again convex (see Rockafellar, 1970, Theorem 2.1), and that a subset of R^n is convex if and only if it contains all the convex combinations of its elements (see Rockafellar, 1970, Theorem 2.2). A convex combination of the vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ in R^n is a sum of the form $\sum_{i=1}^{m} \lambda_i \mathbf{x}_i$, where the coefficients $\lambda_1, \lambda_2, \ldots, \lambda_m$ are all non-negative and sum to 1. Rockafellar (1970, p.23) provides a definition of a *convex function* using the concept of the epigraph of a function. For our purposes it is sufficient to use the following necessary and sufficient condition for

convexity of a function as a definition: a function $f: R^n \to R$ is called convex if $f(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y})$ for all $\mathbf{x}, \mathbf{y} \in R^n$ and $0 < \lambda < 1$ (see Rockafellar, 1970, Theorem 4.1).

2. Results regarding convex optimisation

Fitting a support vector machine requires a convex optimisation problem to be solved. Aspects of this branch of optimisation theory are therefore important when support vector machines are studied. In this section we briefly review the required results.

The problem of finding the value(s) at which a function $f: R^n \to R$ attains a minimum is greatly simplified if f is a convex function. In fact, under fairly mild conditions there is a unique point at which a convex function reaches a minimum. If convexity does not hold, there may be multiple points at which the function under consideration reaches a local minimum, and this makes global minimisation of the function considerably more difficult. Often we have to perform the optimisation subject to certain constraints; that is, we have to solve a constrained optimisation problem. The constraints define the *feasible region*, i.e. the region that has to be searched for optimum points. If each constraint implies a convex region that has to be investigated, we conclude that the feasible region will once again be convex (since the intersection of an arbitrary collection of convex sets is once again convex). Minimising a convex function over a convex region in R^n is called *convex programming*, and this is a simple problem with a unique solution. Fortunately, this

is exactly the type of optimisation that has to be done when fitting a support vector machine.

The general form of the optimisation problems encountered in the theory of support vector machines is therefore as follows:

Minimise a convex objective function $f: R^n \to R$, subject to the inequality constraints $c_i(\mathbf{x}) \leq 0, i = 1,2,\ldots,m$, where each c_i is a convex function, and the equality constraints $e_i(\mathbf{x}) = 0, i = m+1, m+2,\ldots,p$.

The theory for solving such convex optimisation problems is well developed – see for example Rockafellar (1970, Section 28), and Kroon (2003, Appendix A). We form the so-called *primal Lagrangian* by making use of non-negative Lagrange multipliers $\lambda_1, \lambda_2, \ldots, \lambda_p$: $L_P(\mathbf{x},\boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i c_i(\mathbf{x}) + \sum_{i=m+1}^{p} \lambda_i e_i(\mathbf{x})$. The Lagrangian has to be minimised with respect to $\mathbf{x}$ and maximised with respect to the Lagrange multipliers in the vector $\boldsymbol{\lambda}$. A vector pair $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$ is said to be a *saddle point* of L_P (with respect to minimising in $\mathbf{x}$ and maximising in $\boldsymbol{\lambda}$) if

$$L_P(\bar{\mathbf{x}},\boldsymbol{\lambda}) \leq L_P(\bar{\mathbf{x}},\bar{\boldsymbol{\lambda}}) \leq L_P(\mathbf{x},\bar{\boldsymbol{\lambda}})$$

for all $\mathbf{x}$ and $\boldsymbol{\lambda}$. The point $\bar{\mathbf{x}}$ solves the optimisation problem above if and only if $(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}})$ is a *saddle point* of L_P (see Rockafellar, 1970, Theorem 28.3). Necessary and sufficient conditions for this to be the case are the so-called *Karush-Kuhn-Tucker* conditions:

(a) $\bar{\lambda}_i \geq 0$, $c_i(\bar{\mathbf{x}}) \leq 0$ and $\bar{\lambda}_i c_i(\bar{\mathbf{x}}) = 0$ for $i = 1,2,\ldots,m$

(b) $e_i(\bar{\mathbf{x}}) = 0$ for $i = m+1, m+2, \ldots, p$

(c) The derivative with respect to $\mathbf{x}$ of $L_P(\mathbf{x}, \bar{\boldsymbol{\lambda}})$ vanishes at $\mathbf{x} = \bar{\mathbf{x}}$.

How do we go about finding a saddle point of the Lagrangian? An elegant method to achieve this makes use of the concept of *duality*. We briefly explain this in the case where the constraint functions are linear functions of $\mathbf{x}$, and the objective function f is a quadratic function. In this special case the optimisation problem can be solved as a *quadratic programming* problem. The constraints can now be written as:

$$c_i(\mathbf{x}) = c_{i0} + \sum_{j=1}^{n} c_{ij} x_j \leq 0, \; i = 1, 2, \ldots, m$$

$$e_i(\mathbf{x}) = e_{i0} + \sum_{j=1}^{n} e_{ij} x_j = 0, \; i = m+1, m+2, \ldots, p,$$

while the objective function becomes

$$f(\mathbf{x}) = f_0 + \sum_{j=1}^{n} f_j x_j + \sum_{j=1}^{n} \sum_{k=1}^{n} f_{jk} x_j x_k .$$

The so-called Wolfe dual of the primal problem is obtained by putting the derivative of the primal Lagrangian L_P with respect to each of the $x_1, x_2, \ldots, x_n$ equal to zero. This gives:

$$\frac{\partial L_P}{\partial x_j}(\mathbf{x}, \boldsymbol{\lambda}) = f_j + \sum_{k=1}^{n} (f_{jk} + f_{kj}) x_k + \sum_{i=1}^{m} \lambda_i c_{ij} + \sum_{i=m+1}^{p} \lambda_i e_{ij} = 0$$

for $i = 1, 2, \ldots, n$. These are n equations in the n unknowns $x_1, x_2, \ldots, x_n$. We now substitute the constraints implied by these equations back into L_P and the original constraints c_i and e_i where possible. This makes it possible to eliminate some of the primal variables $x_1, x_2, \ldots, x_n$ from L_P and the original constraints. The resulting Lagrangian function is called the dual Lagrangian, L_D. The Wolfe dual problem is now to maximise L_D subject to $\boldsymbol{\lambda} \geq \mathbf{0}$, the constraints which were not used to obtain L_D from L_P, and the modified constraints obtained after substitution.

As an illustration of the above general discussion, consider the following simple example from Kroon (2003, Appendix A). Let the primal problem P be to minimise $f(x, y) = xy$ subject to the constraints $x = y$ and $x \geq 3$. This is clearly equivalent to minimising x^2, subject to $x \geq 3$. The optimal solution is easy to see: $(x, y) = (3, 3)$, with $f(3, 3) = 9$. How can this solution be obtained by using Lagrange multipliers? The primal Lagrangian is given by

$$L_P(x, y, \lambda_1, \lambda_2) = xy + \lambda_1(3 - x) + \lambda_2(x - y).$$

To compute the Wolfe dual, we find the partial derivatives $\frac{\partial L_P}{\partial x}(x, y, \lambda_1, \lambda_2) = y - \lambda_1 + \lambda_2 = 0$, and $\frac{\partial L_P}{\partial y}(x, y, \lambda_1, \lambda_2) = x - \lambda_2 = 0$. These two equations imply that $y = \lambda_1 - \lambda_2$, and $x = \lambda_2$. Substituting these equations into L_P, we obtain after simplification the dual Lagrangian $L_D(\lambda_1, \lambda_2) = \lambda_1^2 - \lambda_1\lambda_2 + 3\lambda_2$. This has to be maximised subject to the constraints $\lambda_1 \geq 3$, $\lambda_2 \geq 0$ and $\lambda_2 = 2\lambda_1$.

In this case the dual problem does not seem to be any simpler than the original problem, but for more complex problems this is often the case. The dual problem can be solved by substituting $\lambda_2 = 2\lambda_1$, yielding the problem of maximising $L_D(\lambda_1) = -\lambda_1^2 + 6\lambda_2$, subject to $2\lambda_1 \geq 0$ and $\lambda_1 \geq 3$. The optimal solution is easily found to be $\lambda_1 = 3$, where $L_D(\lambda_1) = 9$, the optimal value of $f(x, y)$. In addition, $x = \lambda_1 = 3$ and $y = \lambda_2 - \lambda_1 = 2\lambda_1 - \lambda_1 = 3$.

The example illustrates the procedure that is essentially followed in the SVM methodology when partial derivatives are set to 0, and the resulting formulae substituted back into the primal Lagrangian.

Appendix B

R programs

1. R program for classification

```
# This program randomly divides a data set into a training and test sample. It then fits a
# linear discriminant model, a support vector machine as well as a classification tree
# model to the training data, and then uses these models to predict the class membership
# for the test data. The prediction errors are then calculated as the misclassification error
# rate of each model; that is, the percentage of misclassifications. 100 simulations are
# performed, and the average prediction error is calculated for each model.
#
function(Ntimes,Dataset,Fract,Var,Costpar,Gam){
#
# The parameters have the following meaning:
# Ntimes: the number of times the data set will be divided into a training and test sample
# Dataset: the data set that will be used (it is assumed that the response variable is in
# column 1, and that the data is ordered according to the response variable)
# Fract: the proportion of the data set that will be used as training data
# Var: a column vector indicating the variables that will be used
# Costpar: the cost parameter of the support vector machine
# Gam: the gamma (γ) parameter of the SVM [used in the case of the RBF kernel]
#
# The packages needed for the different techniques are now loaded:
#
library(e1071)                  # contains the SVM routine
library(tree)                   # contains the tree routine
library(MASS)                   # contains the linear discriminant analysis routine
#
frame1<-Dataset[,Var]           # the data set is reduced to include only the required
```

```
                                    # variables
attach(frame1)
n<-nrow(frame1)                     # determine the number of rows in frame1
k<-length(frame1)                   # determine the number of variables in frame1
y<-frame1$lapsed                    # y is set equal to the response variable
n2<-sum(y[y==1]); n1<-n-n2          # n2 is calculated as the number of lapse cases, with
                                    # n1 equal to the number of non-lapse cases
#
# The simulation loop starts here
#
fault1<-NULL; fault2<-NULL; fault3<-NULL
for (jj in 1:Ntimes) {
 index1<-sample(1:n1,trunc(n1*Fract))          # a random training sample is drawn
 index2<-sample((n1+1):n,trunc(n2*Fract))      # for lapse and non-lapse cases
 index<-c(index1,index2)
 train<-frame1[index,]                         # the training set is constructed
 test<-frame1[-index,]                         # the test set is constructed
 # now the number of lapse and non-lapse cases in the training and test sets are
 # determined
 ntrain2<-sum(train[,1][train[,1]==1]); ntrain1<-nrow(train)-ntrain2
 ntest2<-sum(test[,1][test[,1]==1]); ntest1<-nrow(test)-ntest2
#
# An ordinary linear discriminant analysis model is fitted to the training data, and the
# result is used to predict the test data. The prediction error is calculated for these
# predictions, and the result is stored.
#
 lda1<-lda(as.factor(train$lapsed)~.,data=train,prior=c(1,1)/2)
 pred1<-predict(lda1,test[,-1])
 v1<-as.numeric(pred1$class)
 v2<-as.numeric(test[,1])+1
 sumv11<-sum(v1[v1==1])
 sumv12<-sum(v1[v1==2])/2
 diffrnce<-v1-v2
 t12<-abs(sum(diffrnce[diffrnce==-1]))
 t21<-sum(diffrnce[diffrnce==1])
 t11<-sumv11-t12
 t22<-sumv12-t21
 ta11[jj]<-t11
 ta12[jj]<-t12
 ta13[jj]<-t21
 ta14[jj]<-t22
 ft1<-sum(abs(as.numeric(test[,1])+1-as.numeric(pred1$class)))/(nrow(test))
 fautl1[jj]<-ft1
#
# A support vector machine is fitted to the training data, and the result is used to predict
# the test data. The prediction error is calculated for these predictions, and the result is
```

```
# stored.
#
 svm1<-svm(as.factor(train$lapsed)~.,data=train,scale=TRUE,shrink=TRUE,
 kernel="radial",cost=Costpar,gamma=Gam,cachesize=500,tolerance=0.00000001)
 # This model can be adjusted as required; for example, to implement a second-degree
 # polynomial kernel, the kernel option will be set to "polynomial" and additional
 # options "degree" and "coef0" will need to be specified for the degree and coefficient
 # parameters of the polynomial kernel.
 pred2<-predict(svm1,test[,-1],type="class")
 tabel<-table(pred=pred2,true=test[,1])
 ta21[jj]<-tabel[1,1]
 ta22[jj]<-tabel[1,2]
 ta23[jj]<-tabel[2,1]
 ta24[jj]<-tabel[2,2]
 ft2 <- (tabel[1,2]+tabel[2,1])/(nrow(test))
 fault2[jj]<-ft2
#
# A classification tree model is fitted to the training data, and the result is used to predict
# the test data.  The prediction error is calculated for these predictions, and the result is
# stored.
#
 tree1<-tree(as.factor(train$lapsed)~.,data=train,split="deviance")
 pred4<-predict(tree1,test[,-1],type="class")
 tabel<-table(pred=pred4,true=test[,1])
 ta31[jj]<-tabel[1,1]
 ta32[jj]<-tabel[1,2]
 ta33[jj]<-tabel[2,1]
 ta34[jj]<-tabel[2,2]
 ft3 <- (tabel[1,2]+tabel[2,1])/(nrow(test))
 fault3[jj]<-ft3

}
 faultavg<-list(sum(fout1)/Ntimes,sum(fault2)/Ntimes,sum(fault3)/Ntimes
 fault1[Ntimes+1]<-sum(fault1)/Ntimes
 fault2[Ntimes+1]<-sum(fault2)/Ntimes
 fault3[Ntimes+1]<-sum(fault3)/Ntimes
 faultfin<-list(fault1,fault2,fault3)
 ta1<-list(ta11,ta12,ta13,ta14)
 ta2<-list(ta21,ta22,ta23,ta24)
 ta3<-list(ta31,ta32,ta33,ta34)
}
# The prediction errors can now be written to files or viewed on screen as required.
```

2. R program for regression

```
# This program randomly divides a data set into a training and test sample. It then fits a
# linear model, a support vector machine as well as a regression tree model to the training
# data, and then uses these models to predict the response variable values for the test data.
# The prediction errors are then calculated as the sum of the squared differences between
# the actual values and the predicted values. A number of simulations are performed, and
# the average prediction error is calculated for each model.
#
function(Ntimes,Dataset,Fract,Var,Costpar,Gam,aeps){
#
# The parameters have the following meaning:
# Ntimes: the number of times the data set will be divided into a training and test sample
# Dataset: the data set that will be used (it is assumed that the response variable is in
# column 1)
# Fract: the proportion of the data set that will be used as training data
# Var: a column vector indicating the variables that will be used
# Costpar: the cost parameter of the support vector machine
# Gam: the gamma (γ) parameter of the SVM [used in the case of the RBF kernel]
# aeps: the epsilon (ε) parameter for epsilon support vector regression.
#
# The packages needed for the different techniques are now loaded:
#
library(e1071)
library(tree)
#
frame1<-Dataset[,Var]          # the data set is reduced to include only the required
                               # variables
attach(frame1)
n<-nrow(frame1)                # determine the number of rows in frame1
#
# The simulation loop starts here
#
fault1<-NULL; fault2<-NULL; fault3<-NULL
for (jj in 1:Ntimes) {
 index<-sample(1:n,trunc(n*Fract))        # a random training sample is drawn
 train<-frame1[index,]                    # the training set is constructed
 test<-frame1[-index,]                    # the test set is constructed
 ntrain<-nrow(train)           # the number of rows in the training sample as well
 ntest<-nrow(test)             # as the test sample is determined
#
# An ordinary linear model is fitted to the training data, and the result is used to predict
# the test data. The prediction error is calculated for these predictions, and the result is
# stored.
#
```

```
 linmod1<-lm(train$TOT.INC~.,data=train)
 pred1<-predict(linmod1,test[,-1])
 ft1<-sum((test[,1]-pred1[])*(test[,1]-pred1[]))/(nrow(test))
 fault1[jj]<-ft1
#
# A support vector machine is fitted to the training data, and the result is used to predict
# the test data.  The prediction error is calculated for these predictions, and the result is
# stored.
#
 svm1<-svm(train$TOT.INC~.,data=train,scale=TRUE,shrink=TRUE,
 type="eps-regression",epsilon=aeps,kernel="radial",cost=Costpar,gamma=Gam,
 cachesize=500,tolerance=0.0000000001)
# An additional SVM parameter that can be adjusted, is the regression type.  In this case
# epsilon regression is used; the regression type could also be specified as "nu-
# regression".
 pred2<-predict(svm1,test[,-1])
 ft2<-sum((test[,1]-pred2[])*(test[,1]-pred2[]))/(nrow(test))
 fault2[jj]<-ft2
#
# A regression tree model is fitted to the training data, and the result is used to predict
# the test data.  The prediction error is calculated for these predictions, and the result is
# stored.
#
 tree1<-tree(train$TOT.INC~.,data=train,split="deviance")
 pred3<-predict(tree1,test[,-1])
 ft3<-sum((test[,1]-pred3[])*(test[,1]-pred3[]))/(nrow(test))
 fault3[jj]<-ft3

}
 faultavg<-list(sum(fout1)/Ntimes,sum(fault2)/Ntimes,sum(fault3)/Ntimes
 faultfin<-list(fault1,fault2,fault3)
}
# The prediction errors can now be written to files or viewed on screen as required.
```

REFERENCES

Banks, D. (1996). A conversation with I.J. Good. *Statistical Science,* **11**, 1-19.

Barnett, V. (1973). *Comparative Statistical Inference.* John Wiley and Sons, New York.

Bennett, K.P. and Campbell, C. (2000). Support vector machines: Hype or hallelujah? *SIGKDD Explorations,* **2**(2). http://www.acm.org/sigs/sigkdd/explorations/issue2-2/bennett.pdf.

Berry, M.J.A. and Linoff, G. (2000). *Mastering Data Mining: The Art and Science of Customer Relationship Management.* John Wiley & Sons, Inc.

Bishop, C.M. (1995). *Neural Networks for Pattern Recognition.* Oxford University Press.

Boser, B.E., Guyon, I.M. and Vapnik, V.N. (1992). A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT.* Pittsburg, PA, ACM Press.

Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984). *Classification and Regression Trees.* Chapman and Hall.

Burges, C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery,* **2**, 121 – 167.

Chang, C.-C. and Lin C.-J. (2001). Training ν-support vector classifiers: Theory and algorithms. *Neural Computation,* **13(9)**, 2119 – 2147.

Collobert, R. and Bengio, S. (2001). SVMTorch: Support Vector Machines for Large-Scale Regression Problems. *Journal of Machine Learning Research*, **1**, 143 – 160.

Cortes, C. and Vapnik, V.N. (1995). Support vector networks. *Machine Learning*, **20**, 273 – 297.

Crammer, K. and Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, **2**, 265 – 292.

Cristianini, N. and Shawe-Taylor, J. (2001). *An Introduction to Support Vector Machines and other Kernel-based Learning Methods.* Cambridge University Press. http://www.support-vector.net.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems,* **2**, 303 – 314.

Drucker, H., Burges, C.J.C., Kaufman, L., Smola, A. and Vapnik, V. (1997). Support vector regression machines. In Mozer, M., Jordan, M.I. and Petsche, T., editors, *Advances in Neural Information Processing Systems,* **9**, 155 – 161. Cambridge, MA, MIT Press.

The Electronic Statistics Textbook. StatSoft Inc.: (http://www.statsoftinc.com/textbook/stathome.html).

Fatti, L.P., Hawkins, D.M. and Raath, E.L. (1982). Discriminant Analysis. In Hawkins, D.M., editors, *Topics in Applied Multivariate Analysis,* Cambridge University Press, Cambridge.

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

Flournoy, N. (1993). A conversation with W.J. Dixon. *Statistical Science,* **8**, 458-477.

Grandvalet, Y. and Canu, S. (2002). Adaptive scaling for feature selection in SVMs. *Advances in Neural Information Processing Systems,* **15**.

Guermeur, Y., Elisseeff, A. and Paugam-Moisy, H. (2000). A new multi-class SVM based on a uniform convergence result. In Amari, S.-I., Giles, C.L., Gori, M. and Piuri, V., editors, *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN 2000).*

Gunn, S.R. (1998). *Support Vector Machines for Classification and Regression.* Technical Report, Faculty of Engineering and Applied Science, Department of Electronics and Computer Science, University of Southampton.

Hair, J.F. Jr., Anderson, R.E., Tatham, R.L. and Black, W.C. (1995). *Multivariate data analysis with readings*. Fourth edition. Prentice Hall.

Hastie, T., Tibshirani, R. and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

Herbrich, R. (2001). *Learning kernel classifiers*. MIT Press, London.

Hertz, J., Krogh, A. and Palmer, R.G. (1991). *Introduction to the theory of Neural Computation*. Lecture Notes Volume I. Addison-Wesley Publishing Company.

Hsu, C.-W., Chang, C.-C., and Lin, C.-J. (2003). *A Practical Guide to Support Vector Classification.* Department of Computer Science and Information Engineering, National Taiwan University.

Hsu, C.-W. and Lin, C.-J. (2001). *A Comparison on Methods for Multi-class Support Vector Machines*. Department of Computer Science and Information Engineering, National Taiwan University.

Kass, G. (1980). An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, **29**, 119 – 127.

Kroon, R.S. (2003). *Generalization Bounds and Model Selection for Transductive Support Vector Machines.* Unpublished M.Com. thesis, Department of Computer Science, University of Stellenbosch.

Laird, N. (1989). A conversation with F.N. David. *Statistical Science,* **4**, 235-246.

Lindsay, B., Kettenring, J. and Siegmund, D. (editors). (2003). *Statistics: Challenges and Opportunities for the Twenty-First Century.* Edition of June 20, 2003.

Mardia, K.V., Kent, J.T. and Bibby, J.M. (1979). *Multivariate Analysis.* Academic Press.

Marquardt, D.W. and Snee, R.D. (1975). Ridge regression in practice. *The American Statistician*, **29**, 3 – 20.

Mendenhall, W. and Sincich, T. (1996). *A second course in Statistics: Regression Analysis.* Fifth edition. Prentice-Hall.

Meyer, D. (2002). *Support Vector Machines: The Interface to libsvm in package e1071.* Technische Universität Wien, Austria.

Milley, A. (2003). Introducing New Mining Capabilities. *sas.com magazine*, **4**.

Morgan, J.N. and Messenger, R.C. (1973). *THAID: A Sequential Search Program for the Analysis of Nominal Scale Dependent Variables.* Survey Research Center, Institute for Social Research, University of Michigan.

Morgan, J.N. and Sonquist, J.A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, **58**, 415 – 434.

Muirhead, R.J. (1982). *Aspects of Multivariate Statistical Theory*. John Wiley & Sons.

Pérez-Cruz, F., Alarcón-Diana, P.L., Navia-Vázquez, A. and Artés-Rodríquez, A. (2001). Fast training of support vector classifiers. In T.K. Leen, T.G. Dietterich, and V. Trest, editors, *Advances in Neural Information Processing Systems*, **13**. MIT Press.

Pérez-Cruz, F. and Artés-Rodríquez, A. (2001). A new optimizing procedure for ν-support vector regressors. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing.*

Potts, W.J.E. (1998). *Data Mining Primer: Overview of Applications and Methods*. Course Notes: SAS Official Curriculum. SAS Institute Inc., Cary, NC, USA.

R Development Core Team. (2003). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.r-project.org.

Raftery, A.E., Tanner, M.A. and Wells, M.T., editors. (2002). *Statistics in the 21st Century*. Monographs on Statistics and Applied Probability, **93**. Chapman & Hall / CRC.

Rockafellar, R.T. (1970). *Convex Analysis*. Princeton Mathematical Series. Princeton University Press, Princeton, New Jersey.

SAS Enterprise Miner Release 4.1. (2000). *Software programme and its documentation.* SAS Institute Inc., Cary, NC, USA.

Schölkopf, B. and Smola, A.J. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

Searle, S.R. (1971). *Linear Models*. John Wiley & Sons.

Smola, A. and Schölkopf, B. (2001). A tutorial on support vector regression. *Statistics and Computing*.

Stecking, R. and Schebesch, K.B. (2003). Support Vector Machines for Credit Scoring: Comparing to and Combining With Some Traditional Classification Methods. In M. Schader, W. Gaul and M. Vichi (Eds.), *Between Data Science and Applied Data Analysis*. Springer, Berlin, 604 – 612.

Sugiyama, M. and Müller, K.-R. (2002). The subspace information criterion for infinite dimensional hypothesis spaces. *Journal of Machine Learning Research*, **3**, 323 – 359.

Swingler, K. (1996). *Applying neural networks: A Practical Guide*. Academic Press.

Taha, H.A. (1997). *Operations Research: An introduction*. Sixth Edition. Prentice-Hall.

Tipping, M.E. (2000). The relevance vector machine. In S.A. Solla, T.K. Leen, and K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems 12*, Cambridge, MA, 652 – 658, MIT Press.

Tipping, M.E. (2001). Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, **1**, 211 – 244.

Vapnik, V.N. (1995). *The Nature of Statistical Learning Theory*. Springer, New York.

Vapnik, V.N. and Chervonenkis, A. (1964). A note on one class of perceptrons. *Automation and Remote Control*, **25**.

Vapnik, V.N., Golowich, S.E. and Smola, A.J. (1997). Support vector method for function approximation, regression estimation, and signal processing. In Mozer, M., Jordan, M.I. and Petsche, T., editors, *Advances in Neural Information Processing Systems*, **9**, 281 – 287. Cambridge, MA, MIT Press.

Vapnik, V.N. and Lerner, A. (1963). Pattern recognition using generalized portrait method. *Automation and Remote Control*, **24**.

Venables, W.N. and Ripley, B.D. (1994). *Modern Applied Statistics with S-Plus*. Springer.

Warner, B. and Misra, M. (1996). Understanding Neural Networks as Statistical Tools. *The American Statistician*, **50(4)**.

Webb, A. (2002). *Statistical Pattern Recognition*. Second edition. Wiley.

Weston, J. & Watkins, C. (1998). *Multi-class Support Vector Machines*. Technical Report. CSD-TR-98-04. Department of Computer Science, Royal Holloway University of London.

Zhu, J. and Hastie, T. (2001). Kernel logistic regression and the import vector machine. Refereed paper accepted for *NIPS2001 Conference*, Vancouver.