Statistical classification in high-dimensional scenarios with a focus on microarray data sets

Tessa Louise Rodseth

Report presented in partial fulfilment
of the requirements for the degree of
MCom **Statistics**
at the University of Stellenbosch

Supervisor: Professor Sarel Steel

December 2017

PLAGIARISM DECLARATION

1. Plagiarism is the use of ideas, material and other intellectual property of another's work and to present it as my own.
2. I agree that plagiarism is a punishable offence because it constitutes theft.
3. I also understand that direct translations are plagiarism.
4. Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.
5. I declare that the work contained in this assignment, except otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

	T. L. Rodseth
Student number	**Signature**
T. L. Rodseth	December 2017
Initials and surname	**Date**

Acknowledgements

I would like to express my deepest gratitude to my supervisor Prof. S.J. Steel for his constant guidance, advice, mentorship and his meticulous attention to detail.

I would like to convey my sincere appreciation to the South African Statistical Association and the National Research Foundation, for the generous funds and financial support they have provided for my MComm and for their contribution towards my future in Statistics.

It has been an honour to study at Stellenbosch University. I would like to thank all the members of staff in the Actuarial Science and Statistics department for providing me with a vast amount of statistical knowledge and encouragement over the past six years.

Finally, special recognition goes to my parents and family for their unconditional love, encouragement and support.

Abstract

High-dimensional data analysis characterises many contemporary problems in statistics and arise in many application areas. This thesis focuses on very high-dimensional problems in which the input predictor variables are gene expression measurements in microarray studies. Accurate analysis of microarray data sets can provide new insight into cancer diagnosis using gene expression profiles and can result in breakthroughs in medical research.

K-nearest neighbours (KNN), fastKNN, linear discriminant analysis (and variants thereof), nearest shrunken centroids (NSC) and support vector machines (SVMs) are investigated in this thesis as binary (and multi-class) classification procedures on microarray data sets.

The important problem of eliminating redundant input variables before implementing classification procedures in high-dimensional data sets is addressed in this thesis. Several variable selection and dimension reduction procedures suitable for microarray data sets are discussed, with the focus on implementing sure independence techniques, NSC and fastKNN feature engineering in the empirical study. Principal component analysis and supervised principal component analysis are implemented as the two main dimension reduction techniques in this thesis.

The performance of the classification procedures is evaluated on three real and three synthetic high-dimensional microarray data sets. The comparison of the different classification methods in the empirical study led to the conclusion that SVMs prove to be the most accurate procedure on the binary data sets considered, whilst NSC is the most accurate procedure on the multi-class data set.

Key words: Classification, K-nearest neighbours, Linear discriminant analysis (LDA), Nearest shrunken centroids (NSC), Support vector machines (SVMs), Variable selection, Dimension Reduction, Principal Component Analysis, Supervised Principal Component Analysis.

Opsomming

Hoë-dimensionele data ontledings is in die huidige tydperk kenmerkend van baie praktiese statistiek probleme. In hierdie tesis is die fokus op hoë-dimensionele data met die onafhanklike veranderlikes wat genetiese metings verteenwoordig, tipies van mikro-skyfie studies. Noukeurige ontleding van mikro-skyfie data kan lei tot nuwe insig in byvoorbeeld die diagnose van kanker waar daar van genetiese profieldata gebruik gemaak word. Dit kan uiteraard tot deurbrake in mediese navorsing lei.

Die KNN tegniek, die sogenaamde "fastKNN" tegniek, lineêre diskriminantanalise (en variasies daarvan), naaste gekrimpte sentroïedes (NSC) en ondersteuningspunt algoritmes (SVMs) word in hierdie tesis ondersoek as klassifikasie prosedures vir binêre en multi-klas mikro-skyfie probleme.

Die belangrike probleem om oortollige en irrelevante veranderlikes uit 'n hoë-dimensionele datastel te elimineer alvorens 'n klassifikasie prosedure daarop toegepas word, word in hierdie tesis aangespreek. Verskeie veranderlike seleksie en dimensie-reduksie prosedures wat geskik is vir toepassing op mikro-skyfie datastelle word bespreek, met die fokus wat geplaas word op "sure independence screening", NSC en "fastKNN". Dit verkry veral aandag in die empiriese gedeelte van die studie. Hoofkomponent analise en gerigte hoofkomponent analise word verder as twee van die vernaamste dimensie-reduksie tegnieke in hierdie tesis geïmplementeer.

Die gehalte van die klassifikasie prosedures word op drie werklike en drie sintetiese hoë-dimensionele datastelle ge-evalueer. Onderlinge vergelyking van die prosedures in die empiriese studie lei tot die gevolgtrekking dat SVMs die akkuraatste prosedure vir binêre datastelle is, terwyl NSC die akkuraatste prosedure vir die multi-klas datastel was.

Sleutelwoorde: Klassifikasie, K-naaste bure, Lineêre diskriminantanalise (LDA), Naaste gekrimpte sentroïedes (NSC), Ondersteuningspunt algoritmes (SVMs), Veranderlike seleksie, Dimensie reduksie, Hoofkomponent analise, Gerigte hoofkomponent analise.

Table of contents

List of figures

List of tables

List of abbreviations and/or acronyms

ALL	Acute Lymphoblastic Leukemia
AML	Acute Myeloid Leukemia
BL	Burkitt Lymphoma
cDNA	Complementary DNA
CV	Cross-validation
DLDA	Diagonal Linear Discriminant Analysis
DNA	Deoxyribonucleic Acid
EWS	Ewing Family of Tumors
FNR	False Negative Rate
KNN	K-Nearest Neighbour
LAR	Least Angle Regression
LASSO	Least Absolute Shrinkage and Selection Operator
LDA	Linear Discriminant Analysis
LOOCV	Leave-one-out Cross-validation
mRNA	Messenger Ribonucleic Acid
NB	Neuroblastoma
NSC	Nearest Shrunken Centroids
PCA	Principal Component Analysis
PLS	Partial Least Squares
RBF	Radial Basis Function
RFE	Recursive Feature Elimination
RMS	Rhabdomyosarcoma
RNA	Ribonucleic Acid
RSS	Residual Sum of Squares
SIS	Sure Independence Screening

SPCA	Supervised Principal Component Analysis
SRBCT	Small round blue cell tumour
SVD	Singular Value Decomposition
SVM	Support Vector Machine
VS	Variable selection

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

Consider a data set consisting of N observations on each of p variables. In earlier times, almost all data sets had N larger than p, and most traditional statistical procedures were designed for scenarios where this requirement held true. Mainly as a consequence of the growth in computing power, the last approximately fifteen years have seen a proliferation of data sets where the number of variables exceeds the number of data cases, sometimes by a large factor. Such data sets (referred to as *wide* data sets because of the appearance of the number of data cases by number of variables data matrix) are now well known in genomics, but also occur in fields such as econometrics and near-infrared spectroscopy. Analysing such high-dimensional data sets is currently an important problem in statistics. This thesis focuses on one high-dimensional problem in which the predictor variables are gene expression measurements in microarray studies. Class prediction in microarray problems is important and has recently received a great deal of attention. The main task is to classify and predict the diagnostic category of a sample, based on its gene expression profile.

Although regression and classification procedures have been developed to deal directly with wide data sets, variable selection and dimension reduction remains an important tool in their analysis. This thesis discusses several standard statistical classification procedures, as well as classification procedures that are not well established in traditional statistics. The former group includes the nearest neighbour classifier, (linear) discriminant analysis and one of its variants. The *perceptron algorithm, optimal separating hyperplanes*, *support vector machines* (SVMs), *nearest shrunken centroids* (NSC) and other techniques that originated in machine learning are included in the latter group. Procedures using *linear discriminant analysis* (LDA) are popular options since they are easy to implement and interpret, and by avoiding the danger of overfitting, they frequently yield classifiers with good generalisation properties. Naturally scenarios arise where cases belonging to distinct classes cannot be separated satisfactorily by a linear discriminant (classification) function, and non-linear computer-intensive classifiers such as SVMs have been proposed for such problems. SVMs form a supervised learning technique that has proven useful in classification problems encountered in working with microarray data.

Regarding statistical dimension reduction, in this thesis it will be viewed as an approach to reduce the number of variables in a problem by replacing them with a much smaller number of transformed versions of the original variables. Typically these transformations are linear combinations of the original variables. Dimension reduction methods for classification can be categorized into supervised and unsupervised methods. *Principal component analysis* (PCA) belongs to the latter group and is the oldest and most popular dimension reduction approach, where the linear combinations are formed to successively maximise variance. *Supervised principal component analysis* (SPCA) and *partial least squares* (PLS) are examples of two supervised dimension reduction methods considered in the thesis.

A related approach, also appropriate for scenarios with a large number of variables, is that of variable selection. In variable selection there is no transformation of the original variables, but simply a reduction in their number by elimination of seemingly irrelevant and redundant variables. Compared to dimension reduction, which uses linear combinations of the original variables, this approach has the advantage of retaining the interpretability of the original variables. Furthermore, variable selection has the advantage of identifying the most important predictor variables. However, because of the discrete nature of variable selection (*i.e.* a variable is either retained or eliminated), as well as the difficulty of deciding on the number of variables to retain, models based on a reduced number of variables do not always turn out to be more accurate than those using all of the variables or those based on (a small number of) principal components.

The different approaches referred to above will be compared in several numerical investigations, comprising simulation as well as application of the methods to different practical data sets from the medical field. In all cases the focus will be on $p > N$ scenarios, and mainly on binary classification.

1.2 NOTATION AND TECHNICAL PRELIMINARIES

Consider the following generic two-group (binary) classification problem. In supervised learning in general, the observed response or dependent variable, Y, can be either quantitative (numerical) or qualitative (categorical). In a binary classification problem Y

is categorical, *i.e.* $Y \in \{0, 1\}$. The input or predictor variables are denoted by $X_1, X_2, \dots, X_p$, where p represents the number of predictor variables. These variables are observed for $N = N_1 + N_2$ sample cases, with the first N_1 cases from Population 1, P_1 (where $Y = 0$) and the remaining N_2 cases from Population 2, P_2. The resulting observed data are represented as $\{(\boldsymbol{x}_i, y_i), i = 1, \dots, N\}$. Here, $\boldsymbol{x}_i$: $p \times 1$ is a vector containing values of $X_1, .., X_p$ for the i^{th} data case, with the corresponding value y_i for the response. Finally, $\boldsymbol{y}$: $N \times 1$ denotes for the vector containing $y_1, \dots, y_N$ and X: $N \times p$ the matrix containing the observations of $X_1, \dots, X_p$.

1.3 LITERATURE REVIEW

The problem of learning in high-dimensional feature spaces arises in diverse fields of scientific research and technological development, including genomic and proteomic studies as well as document classification (Fan and Lv, 2010). The second edition of Hastie *et al.* (2009) includes a new chapter which is devoted to high-dimensional problems, highlighting the importance of $p > N$ applications.

Many procedures have been proposed for dealing with high-dimensional classification problems and these problems have recently received a great deal of attention in the context of gene expressions measurements in DNA microarray studies. The advances in microarray technology have led to a large amount of statistical research: see for example the books by Speed (2003), Parmigiani *et al.* (2003), Simon *et al.* (2004), Lee (2004), and Li and Xu (2008).

For general discussions of classification (not focussing on high-dimensional settings) the reader is referred to the texts by Hastie *et al.* (2009), Mardia *et al.* (1979), McLachlan (1992), and Ripley (1996). Relevant work on statistical aspects of classification in the context of microarray experiments includes: Ambroise and McLachlan (2002), Dettling and Bühlmann (2002), Dudoit *et al.* (2002), Golub *et al.* (1999) and Tibshirani *et al.* (2001). However, comparative studies on microarray data sets are limited, and furthermore Ambroise and McLachlan (2002) report that such comparative studies have not always been properly calibrated. Interested readers are referred to Hastie *et al.* (2009, Chapter 7) for a detailed explanation of the common practice of omitting any selection or filtering steps from the cross-validation process.

The paper by Dudoit *et al.* (2002) provides an overview and comparison of discrimination methods for gene expression data, including the *nearest-neighbour classifier*, *linear discriminant analysis* and *diagonal linear discriminant analysis* (DLDA). The nearest-neighbour classification methods are based on classifying test set observations using the distance function of the learning set. The nearest-neighbour methods goes back at least to Fix and Hodges (1951), and extensive literature on the topic is reviewed by Dasarathy (1991). A detailed discussion of the manifestation of the curse of dimensionality in the nearest-neighbour procedure is given in Hastie *et al.* (2009). Specifically, the authors state that for very sparse neighbourhoods in a high-dimensional space the KNN classifier will lead to high variances in the predictions. The fastKNN procedure refers to a modified version of the KNN procedure proposed by Pinto (2016), which will be investigated in the thesis.

Fisher proposed LDA in 1936 and it has become one of the most frequently used statistical classification techniques for solving binary classification problems. In LDA, the decision boundary is determined by the covariance matrix of the class distributions and the position of the class centroids (mean vectors). LDA is especially successful in cases where $X_1, X_2, \dots, X_p$ have approximate normal distributions, but cannot be applied to high-dimensional problems. Attempts to broaden the applicability of LDA have led to several extensions of the basic discriminant analysis technique, yielding for example quadratic discriminant analysis, flexible discriminant analysis, regularized discriminant analysis and DLDA (see Hastie *et al.*, 2009, for a discussion of these techniques).

DLDA in classification problems is based on the idea of class-wise independence, *i.e.* conditional independence of the predictor variables given the class membership. Levina (2002) performs a mathematical analysis comparing DLDA to full LDA in a high-dimensional setting and illustrates that with reasonable assumptions, DLDA has a lower asymptotic error rate than full LDA. Bickel and Levina (2004) provide some interesting theoretical results for DLDA. Tibshirani *et al.* (2001) and Tibshirani *et al.* (2003) propose the nearest shrunken-centroid classifier that is employed in this thesis. A further contribution is made in the paper by Efron (2008) which proposes an empirical Bayes variation of nearest shrunken centroids.

Chapter 2 closes with an explanation of SVMs based on the discussion presented in Hastie *et al.* (2009, Chapters 4 and 12). SVMs were introduced by Vapnik *et al*. in 1992 and have since been popular due to their sound theoretical background and effective practical applications. They have become commonly used classification tools and are considered to be highly flexible methods, suffering however from poor interpretability. SVMs can be used for classification in cases with linearly separable and non-linearly separable data sets. The concept of optimal separation by hyperplanes is usually introduced via maximal margin classifiers. Support vector classifiers take this idea a step further and include a regularization cost parameter which allows for more flexible and accurate predictions in cases of non-linearly separable data sets. Finally, SVMs are quite powerful due to their effective predictions even for non-linear classification by using the kernel trick.

Turning to variable selection, it should be noted that in many classification problems only a subset of $X_1, X_2, \dots, X_p$ is required for separating the classes, and frequently the use of only these variables for constructing classification functions yields more accurate classifiers. As a result, variable selection plays a fundamental role in classification of high-dimensional data sets. Of the classification methods discussed in this thesis, only fastKNN and NSC perform automatic variable selection. Neither discriminant analysis nor the SVM classifier perform automatic variable selection (as they use quadratic instead of absolute regularisation). In general, the classification accuracy of SVMs can be substantially improved if instead of all p original variables, an appropriate smaller subset of variables is used.

Recent references to the important topic of variable selection in statistics include Fan and Lv (2008), Paul *et al.* (2008), Hastie *et al.* (2009), Fan and Lv (2010), Kulkarni *et al.* (2011) and James *et al.* (2013), while Louw (1997), and references cited therein, provide a comprehensive overview of classification variable selection procedures. This thesis will explore several variable selection techniques discussed in the paper by Fan and Lv (2010) which cope with high-dimensionality. Fan and Lv (2010) is a well-established paper that is cited over 400 times.

The most commonly used selection procedures for gene expression data are based on the use of a score criterion which is calculated for each of the genes individually, and then the genes where this criterion exceeds a pre-specified threshold are selected. Such methods are discussed in Chapter 3. The main advantages of this kind of variable selection are its simplicity and the interpretability of the results; however, such individual gene-wise selection does not provide for interactions and correlations between genes and these are not taken into consideration during the selection.

Modern procedures such as the *least absolute shrinkage and selection operator* (LASSO) (Tibshirani, 1996), forward stagewise regression (Hastie *et al.,* 2001) and *least angle regression* (LAR) (Efron *et al.*, 2004) were developed as a consequence of the fact that many traditional variable selection methods were no longer sufficient in high-dimensional problems. Pre-conditioning (Paul *et al.,* 2008) is an alternative feature selection approach based on finding a consistent predictor $\hat{y}$ for the response (a supervised principal component approach was the initial fitting method used), followed by application of a fitting procedure such as the LASSO to the resulting values.

Dimension reduction methods are explored in the literature as an alternative to variable selection in order to overcome the curse of dimensionality. PCA is a useful unsupervised statistical technique that is often used in the analysis of genomic data and is discussed with reference to the papers by Ghosh (2002) and Rencher (2002). However, the unsupervised nature of PCA implies that it has shortfalls, since it does not take the response variable into account. The paper by Bair *et al.* (2006) proposes supervised principal components, an approach which takes the response variable into account and thereby overcomes some of the problems faced by PCA. This thesis includes a detailed description of supervised principal components with the papers by Bair *et al*. (2006) and Paul *et al*. (2008) as points of departure. PLS dimension reduction, introduced by Wold (1975), is known to give good prediction accuracy, feature selection and visualization in the context of classification problems with high-dimensional microarray data. A brief description of PLS is given later in the thesis with reference to papers by Hastie *et al.* (2009), Chung and Keleş (2010) and Boulesteix (2004).

The study described in this thesis is based on the analysis of three real microarray cancer data sets and three simulated microarray data sets. The results from relevant and top ranking papers which implement different classification procedures on the colon cancer, leukemia and SRBCT data set are used as a benchmark to compare the results of the empirical work presented in this thesis. Obviously one would like to propose a classification method that is superior to all existing methods. Unfortunately, the results reported in Chapter 6 reveal that although some of the classification procedures have come close it has not been possible to improve on existing approaches. Additionally it should be noted that different papers used different splits of data into training and test parts.

The first practical data set which is considered in this thesis is the colon cancer microarray data set which was originally studied by Alon *et al.* (1999) and subsequently investigated by Guyon *et al.* (2002), Weston *et al.* (2003) and Rakotomamonjy (2003). The results of the classification procedures considered in this thesis on the colon cancer data set are compared to the findings in the above-mentioned papers as well as to some of the results in papers by Ben-Dor *et al.* (2000), Boulesteix (2004), Alladi *et al.* (2008) and Kulkarni *et al*. (2011). Genetic information for the gene expressions in the colon data set was found in the papers by Cherian *et al.* (2003), Yap *et Al.* (2004), Jiang *et al.* (2008), Notterman *et al.* (2001), Shailubhai *et al.* (2000) and Kishino *et al.* (2000).

The second data set considered in the thesis is the leukemia data set, which was introduced by Golub *et al.* (1999). The pre-processed leukemia data set described in Dudoit *et al.* (2002) is considered in the empirical study. The paper by Chow *et al.* (2001) also compares the performance of different classification procedures on the leukemia data set.

The SRBCT data set presented in Khan *et al.* (2001) is the third and final real microarray data set considered in this thesis. The SRBCT data set is the only multi-class problem considered in the thesis. The results presented for the SRBCT data are briefly compared to those reported in papers by Tibshirani *et al.* (2001), Khan *et al.* (2001), Tibshirani *et al.* (2002) and Liu *et al.* (2009).

1.4 CHAPTER OUTLINE

This thesis consists of seven chapters, the first of which is the current introductory chapter. The body of the thesis is divided into two parts: the theoretical Chapters 2, 3 and 4, and the chapters describing the empirical work and conclusions, namely Chapters 5, 6 and 7.

Chapter 2 provides a discussion of different binary classification methods, namely k-nearest neighbours, classical linear discriminant analysis, nearest shrunken centroids and support vector machines. Furthermore, various aspects of these classifiers are discussed, including their limitations and modifications. Variable selection is the primary focus of Chapter 3, which contains a brief discussion of the rationale behind variable selection procedures, particularly in high-dimensional settings. A description of three different dimension reduction techniques that can be used in statistical classification is given in Chapter 4. In this chapter the distinction between the dimension reduction techniques is provided, and in certain cases supported with some experimental work. In Chapter 5 a description of the practical and simulated data sets considered in this thesis is given, as well as an outline of the classification procedures that will be applied to the binary and multi-class data sets. Chapter 5 concludes with a report of the empirical study to evaluate the selection performance of different candidate threshold values for the KNN, SVM and NSC classifiers.

Chapter 6 presents the results of the empirical study and compares the performance of the different classification techniques by means of analysis of six high-dimensional classification data sets. This chapter closes by comparing the top ranking classification procedures across all data sets to establish if one procedure appears to be consistently superior to the others in terms of both classification accuracy (using the test error rate) and the number of features used.

Finally, a summary of the main findings and some concluding remarks are given in Chapter 7. This chapter also offers some limitations of the current study and recommendations for future research.

CHAPTER 2
STATISTICAL CLASSIFICATION

2.1 INTRODUCTION

Consider a supervised statistical problem with response variable Y and input variables $X_1, X_2, \ldots, X_p$. In supervised problems, an important general objective is to predict the value of Y from observations of the input variables. If Y is a qualitative (categorical) variable assuming values in the unordered set $\{1,2,\ldots,G\}$, the problem is one of classification. The case $G = 2$ leads to a *binary* classification problem. The focus in this thesis is mainly on binary classification problems.

Traditional classification methods include LDA and logistic regression (LR). Although these methods are still widely used, they break down when applied to wide data sets. This chapter will focus on brief explanations of classification procedures used later in the thesis, namely K-nearest neighbours, LDA, nearest shrunken centroids and SVMs.

2.2 BINARY CLASSIFICATION

This thesis will focus on binary classification and therefore the statistical classification procedures are explained in binary scenarios, where $G = 2$. For multi-class classification settings where $G > 2$, considered in parts of the empirical study, the necessary changes to the statistical classification procedures will be stated explicitly. The remainder of this chapter focuses exclusively on the binary case.

2.3 K-NEAREST NEIGHBOURS

The K-nearest neighbours (KNN) classifier is a conceptually simple non-parametric classification procedure that predicts the class of a test case by a majority vote amongst the K nearest neighbours of the test case. The KNN classifier is an attempt to approximate the Bayes classifier by estimating the conditional distribution of an outcome value given the predictor values. It then classifies a given observation to the class with the highest estimated probability (James *et al.,* 2013:39).

The KNN procedure is described as follows by James *et al.* (2013:39). Given a positive integer K and a test observation, $\boldsymbol{x}_0$, the KNN classifier first identifies the K points in the training data closest to $\boldsymbol{x}_0$, denoted by $\mathcal{N}_0$. Euclidean distance is used to measure the proximity of cases in the training set to the test case, *i.e.*

$$d_{(i)} = \left\|\boldsymbol{x}_{(i)} - \boldsymbol{x}_0\right\|.$$

The K nearest neighbours of $\boldsymbol{x}_0$ are the K training cases with the smallest Euclidean distance to $\boldsymbol{x}_0$. The KNN classifier then estimates the conditional probability for class g as the fraction of points in $\mathcal{N}_0$ whose response values equal g, *i.e.*

$$\Pr(Y_0 = g \mid \boldsymbol{X} = \boldsymbol{x}_0) = \frac{1}{K}\sum_{i\epsilon\mathcal{N}_0} I(y_i = g). \tag{2.1}$$

Finally, KNN applies the Bayes rule and classifies $\boldsymbol{x}_0$ to the class with the largest estimated probability computed in Equation (2.1).

The KNN classifier is drastically influenced by the number of neighbours used in the procedure, *i.e.* the value of K. Consequently, the value of K is typically chosen by cross-validation. A small value of K, for example $K = 1$, results in an overly flexible non-linear decision boundary, and corresponds to a very high variance but low bias classifier (James *et al.,* 2013:40). As the value of K increases, the KNN decision boundary becomes less flexible, which corresponds to a low variance but high bias classifier. In the experimental study in this thesis, three values of K are used, namely the values {1, 3, 5}. Note that only odd values of K are considered to avoid ties when classification is performed.

Although KNN is a very simple approach, it often produces a classifier that is very similar to the optimal Bayes classifier even though the true distribution of the data is unknown (James *et al.,* 2013:39). Hastie *et al.* (2009:22) provide a detailed description of the curse of dimensionality for local methods such as the KNN procedure and state that in high dimensions KNN does not perform well. The results of the experimental study reported in Chapter 5 and 6 interestingly somewhat contradict this statement.

2.3.1 FastKNN

It should first of all be noted that the term “fastKNN” yields two different results when googled: firstly, it refers to a modified version of the KNN procedure proposed by Pinto (2016), and secondly, it refers to the R package *fastknn* developed to rapidly implement the KNN procedure. The discussion in this section deals with the first meaning of *fastKNN*.

The fastKNN classifier is an adaptation of the traditional KNN classifier and was developed to make it easier to create and tune KNN classifiers for very large data sets, *i.e.* cases where $N > 100\,000$. The fastKNN procedure provides shrinkage estimates of the class membership probabilities, based on the reciprocal distances of the nearest neighbours (Pinto, 2016). Consider a test case with input vector $\boldsymbol{x}_0$ and denote the corresponding label by Y_0. Let $\mathcal{N}_0$ index the K nearest neighbours of $\boldsymbol{x}_0$ in the training set. It is of importance to estimate $P(Y_0 = g)$ for $g = 1,2$. According to fastKNN take

$$\mathrm{P}(\mathrm{Y}_0 = g) = \frac{\sum_{i \in \mathcal{N}_0} \frac{1}{d_i} Ind(Y_i = g)}{\sum_{i \in \mathcal{N}_0} \frac{1}{d_i}}.$$

Here d_i is the distance between $\boldsymbol{x}_0$ and its i^{th} nearest-neighbour. In the above equation the denominator normalises this probability. The test case is now classified according to

$$g_0 = \arg \max_{g \in \{1,2\}} P(Y_0 = g).$$

Note that the fastKNN estimator uses a weighted voting rule, as the neighbours close to $\boldsymbol{x}_0$ have more influence on predicting the class of $\boldsymbol{x}_0$ than the neighbours further away from of $\boldsymbol{x}_0$.

Pinto (2016) lists six reasons why the fastKNN classifier was developed:

i. to build a KNN classifier for larger data sets that can be implemented in a few seconds (as the procedure generates new features instead of working with all the variables in the data set)

ii. it uses a weighted estimator which predicts more calibrated probabilities compared to the traditional KNN estimator

iii. it provides an interface to determine the best value of the parameter K according to a variety of loss functions, using N-fold cross validation

iv. it is capable of plotting beautiful classification decision boundaries for the data set

v. it performs feature engineering and extracts highly informative features from the data set

vi. and finally, it improves one's performance in Kaggle competitions.

In this thesis, fastKNN is applied to perform feature engineering on high-dimensional data sets. Feature engineering is based on the ideas presented in the winning solution of the Otto Group Product Classification Challenge on Kaggle. Pinto (2016) explains that fastKNN generates $M = K \times G$ new features, $Z_1, Z_2, \dots, Z_M$, which are then used to classify $\boldsymbol{x}$, where G is the number of groups ($G = 2$ for binary classification) and K is the number of neighbours considered. Note that the choice of K is directly related to the number of new features, so one must be cautious not to select a large value for K in a large data set as it will be computationally expensive. The M new features are computed from the distances between the observations and their K nearest neighbours for each class. There are therefore K new features for each group. Consider a vector $\boldsymbol{x}$ of input variable values. The first new feature for $\boldsymbol{x}$ is computed as the distance between $\boldsymbol{x}$ and the single nearest neighbour in Group 1, the second feature is the sum of the distances from $\boldsymbol{x}$ to the two nearest neighbours in Group 1, and so on, until K new features have been computed; then the $K + 1^{th}$ new feature is computed as the distance from $\boldsymbol{x}$ to the single nearest neighbour in Group 2, and so forth. The following figure illustrates how the fastKNN procedure generates new features on a simulated data set.

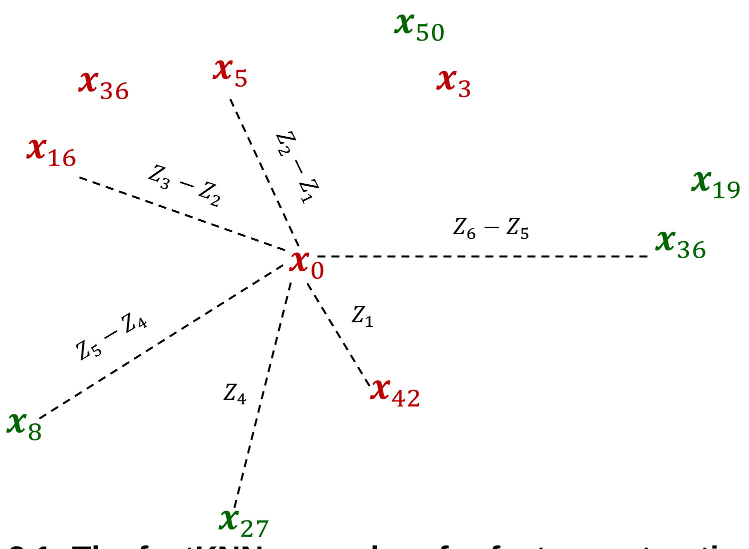

Figure 2.1: The fastKNN procedure for feature extraction

In Figure 2.1, the simulated data set consists of 50 data cases ($N = 50$) belonging to two populations, namely P_1 and P_2, which are represented by red and green labels respectively. The test observation, $\boldsymbol{x}_0$, belongs to P_1 and the five nearest neighbours, in terms of Euclidean distance, for each of the classes are illustrated in Figure 2.1. Feature engineering using fastKNN is applied to the simulated data using $K = 3$ and therefore six $(K \times G = 3 \times 2)$ new features $Z_1, Z_2, \dots, Z_6$ are extracted. The new feature Z_1 measures the distance from the nearest neighbour in P_1, data case 42 ($\boldsymbol{x}_{42}$), to $\boldsymbol{x}_0$. The new feature Z_2 is the sum of the distances between $\boldsymbol{x}_0$ and the two nearest neighbours in P_1, namely $\boldsymbol{x}_{42}$ and $\boldsymbol{x}_5$. Similarly, Z_3 is the sum of the distances from $\boldsymbol{x}_{42}, \boldsymbol{x}_5$ and $\boldsymbol{x}_{16}$ to $\boldsymbol{x}_0$. The values of Z_4, Z_5 and Z_6 are based on the distances to the first, second and third nearest neighbours of $\boldsymbol{x}_0$ in P_2. Therefore, Z_4 measures the distance from the nearest neighbour in $P_2, \boldsymbol{x}_{27},$ to $\boldsymbol{x}_0$, Z_5 is the sum of the distances of $\boldsymbol{x}_{27}$ and $\boldsymbol{x}_8$ to $\boldsymbol{x}_0$ and Z_6 is the sum of the distances from $\boldsymbol{x}_{27}, \boldsymbol{x}_8$ and $\boldsymbol{x}_{36}$ to $\boldsymbol{x}_0$.

The following provides a summary of the algorithm for fastKNN feature extraction applied to the general case of G groups. Consider a training data set $\mathcal{T} = \{\boldsymbol{x}_i, y_i), i = 1, \dots, N\}$. Let $J_1, \dots, J_G$ be a partition of $\{1,2, \dots, N\} = J$, with $J_j = \{i \in J: y_i = g\}, g = 1, \dots, G$. The new features for the test case $\boldsymbol{x}_0$ are constructed as follows:

1. Find $i_g \in J_g$ where $\boldsymbol{x}_{i_g}$ is the closest training data case to $\boldsymbol{x}_0$ with index in $J_g, g = 1, \dots, G$.
2. Compute $z_g = \left\| \boldsymbol{x} - \boldsymbol{x}_{i_g} \right\|, g = 1, \dots, G$.
3. Repeat Steps 1 and 2, but now find the two closest neighbours of $\boldsymbol{x}_0$ in each of the G classes and compute $z_{g+G}, g = 1, \dots, G$, as the sum of the distances between $\boldsymbol{x}_0$ and the two nearest neighbours.
4. Repeat until $z_1, .., z_G, z_{G+1}, \dots, z_{2G}, z_{(K-1)G+1}, \dots, z_{KG}$ have been computed in this manner.

Now a vector $\boldsymbol{z} = [z_1, \dots, z_M]$ of new feature values has been computed for each test case, where $M = KG$.

In order to apply KNN to classify $\boldsymbol{x}_0$ (or $\boldsymbol{z}$) into one of the available classes, each training $\boldsymbol{x}_i \in \mathcal{T}$ also has to be transformed to a corresponding $\boldsymbol{z}_i$. In this feature engineering process, cross-validation (CV) is required, say r- fold CV, on each new training data set to avoid overfitting (Pinto, 2016). So, randomly split the training data into r mutually exclusive folds. Consider the first of these folds. Treat each of the cases in this fold as if it were a test case and compute a vector $\boldsymbol{z}_i$ exactly as described above for every $\boldsymbol{x}_i$ in this first fold. The procedure is repeated for all the folds, thereby obtaining a matrix $Z: N \times M$ $(M = KG)$ that can be used in place of $X: N \times p$. Note that this approach will remove the zeroes that are otherwise (without CV) obtained as part of each of the first G new features. Finally, ordinary KNN or any alternative classifier can be used to classify $\boldsymbol{x}_0$, using $\boldsymbol{z}$ as test observation vector and Z as the training input matrix.

2.4 CLASSICAL LDA

Classical LDA was proposed by Fisher in 1936 and is a frequently used method for statistical classification. In LDA, as in all other fully supervised procedures, it is necessary to know which population the cases in the training sample (initial sample) belong to in order to be able to classify cases whose population memberships are unknown. Furthermore, LDA is a useful technique to determine which variables play an important role in classification of an observation (Afifi and Clark, 1997:244).

LDA can be applied in both a binary and a multi-class classification setup. This thesis will focus on binary LDA, using Fisher's method to discriminate between the two populations. Fisher's approach does not assume normality; however, it does assume that the two populations have the same variance-covariance matrix. For the purpose of this thesis, Fisher's linear discriminant function will be explained in terms of two populations or groups, denoted by P_1 and P_2, as before. The available training data consists of N_1 observations from P_1 and N_2 observations from P_2. The mean of the observations sampled from P_1 is denoted by $\bar{\boldsymbol{x}}_1$, while the mean of the observations sampled from P_2 is denoted by $\bar{\boldsymbol{x}}_2$. The total sample size is $N = N_1 + N_2$.

Fisher's LDA aims to find a subspace yielding maximum separation between the two classes. This optimal space is obtained by maximising the between-class variation, whilst simultaneously minimising the within-class variation. The between-class variation needs to be maximised – the larger the distance between the two populations, the easier it is to distinguish between these groups and accurately classify observations. Additionally, the within-class variation needs to be minimised. If there is a large overlap between the two populations, it will be difficult to distinguish between the two groups, as it may result in a large error when predicting the group of each observation.

Fisher's LDA is implemented by maximising the Rayleigh coefficient, *i.e.* by solving

$$\max_{\boldsymbol{w}} \left\{ \frac{\boldsymbol{w}^T S_b \boldsymbol{w}}{\boldsymbol{w}^T S_w \boldsymbol{w}} \right\}, \text{ subject to } \|\boldsymbol{w}\| = 1. \tag{2.2}$$

Here the between-class covariance matrix, S_b, is calculated as

$$S_b = \frac{1}{N} \sum_{g=1}^{G} N_g (\overline{\boldsymbol{x}}_g - \overline{\boldsymbol{x}})(\overline{\boldsymbol{x}}_g - \overline{\boldsymbol{x}})^T = H_b H_b^T.$$

Similarly, the within-class covariance matrix, S_w, is calculated as

$$S_w = \frac{1}{N} \sum_{g=1}^{G} \sum_{i=1}^{N_i} (\boldsymbol{x}_{gi} - \overline{\boldsymbol{x}}_g)(\boldsymbol{x}_{gi} - \overline{\boldsymbol{x}}_g)^T = H_w H_w^T.$$

In these expressions, $\overline{\boldsymbol{x}}$ represents the overall mean vector for all the data, and $\overline{\boldsymbol{x}}_g$ represents the mean vector for Group g, where $g = 1, .., G$. For the purpose of this study, $G = 2$. The total covariance matrix, S_t, is fixed for the full data set and is the sum of S_w and S_b (Zhang and Sim, 2007). S_b and S_w both have dimension $p \times p$, which is very large in the scenario where there are many variables. To reduce computations, the precursor matrices, H_b and H_w, are often used. The precursor matrix H_b has dimension $p \times G$ and is defined by

$$H_b = \frac{1}{\sqrt{N}} \left[\sqrt{N_1} (\overline{\boldsymbol{x}}_1 - \overline{\boldsymbol{x}}), \dots, \sqrt{N_G} (\overline{\boldsymbol{x}}_G - \overline{\boldsymbol{x}}) \right].$$

The precursor matrix H_w is of dimension $p \times N$ and is defined by

$$H_w = \frac{1}{\sqrt{N}} \left[(\boldsymbol{x}_{11} - \overline{\boldsymbol{x}}_1), \dots, (\boldsymbol{x}_{1N_1} - \overline{\boldsymbol{x}}_1), \dots, (\boldsymbol{x}_{G1} - \overline{\boldsymbol{x}}_G), \dots, (\boldsymbol{x}_{GN_G} - \overline{\boldsymbol{x}}_G) \right].$$

Clearly for this research study $p \gg G$ and $p \gg N$, so the use of these precursor matrices is computationally more efficient.

LDA searches for a projection to transform the p-dimensional input data to a lower dimension. Multi-class LDA uses a projection matrix, but in the case of binary LDA, which is considered in this study, a projection vector $\boldsymbol{w}$ is obtained by solving the maximisation problem in (2.2) above.

To solve the maximisation problem in (2.2), eigen-analysis is implemented on $S_w^{-1} S_b$. Let the non-zero eigenvalues of this matrix be denoted by $\hat{\lambda}_1, \hat{\lambda}_2, \dots \hat{\lambda}_s$, and the corresponding normalised eigenvectors by $\hat{\boldsymbol{e}}_1, \hat{\boldsymbol{e}}_2, \dots \hat{\boldsymbol{e}}_s$ (normalised so that $\hat{\boldsymbol{e}}^T \hat{\boldsymbol{e}} = 1$). The number of non-zero eigenvalues, s, satisfies $1 \leq s \leq \min(G-1, p)$. For binary LDA, $s = G - 1 = 1$. According to Johnson and Wichern (2007:430), the vector that maximises (2.2) is given by $\boldsymbol{w} = \hat{\boldsymbol{e}}_1$, which is the leading eigenvector of $S_w^{-1} S_b$.

Fisher's method then projects the data from p dimensions to s dimensions. In the lower dimensional space, a closest mean (centroid) classifier is used to classify new observations with unknown group memberships. The class centroids in the lower dimensional space are given by $\boldsymbol{w}^T \overline{\boldsymbol{x}}_g$. The midpoint between these transformed group means is calculated as:

$$\tfrac{1}{2}\boldsymbol{w}^T(\overline{\boldsymbol{x}}_1 + \overline{\boldsymbol{x}}_2).$$

To classify a new case, $\boldsymbol{x}_0$, the new case is projected to the lower dimensional space by calculating $y_0 = \boldsymbol{w}^T \boldsymbol{x}_0$. The allocation rule based on Fisher's discriminant function is as follows: allocate an observation, $\boldsymbol{x}_0$, to P_1, if $y_0 - \frac{1}{2}\boldsymbol{w}^T(\overline{\boldsymbol{x}}_1 + \overline{\boldsymbol{x}}_2) \geq 0$. Otherwise allocate $\boldsymbol{x}_0$ to P_2, *i.e.* if $y_0 - \frac{1}{2}\boldsymbol{w}^T(\overline{\boldsymbol{x}}_1 + \overline{\boldsymbol{x}}_2) < 0$.

A fundamental limitation of classical LDA is that the matrix S_w must be non-singular in order for $S_w^{-1} S_b$ to be computed. Therefore LDA cannot be applied to wide data sets, as S_w is singular; this is known as the singularity problem (Ye and Li, 2005:929). Two advantages of LDA are its simplicity and the possibility of extending it in various ways. An extension which overcomes the singularity problem is so-called diagonal LDA (DLDA), which is discussed next.

2.4.1 Diagonal LDA

DLDA solves the problem of S_w being singular when fitting a full LDA in a high-dimensional setting by making use of a regularisation assumption. DLDA makes the simple assumption that the predictor variables are independent within each class, which implies that the within-class covariance matrix, which is ordinarily non-diagonal, now becomes diagonal. Together with the assumption of equal variances for the predictor variables within the different classes, this implies that for all population groups

$$S_w = diag\left(\sigma_{g1}^2, \dots, \sigma_{gp}^2\right),$$

i.e. S_w is a $p \times p$ diagonal matrix. The assumption made in DLDA is equivalent to the assumption made in the naïve-Bayes classifier, also known as idiot's Bayes. The naïve-

Bayes classifier assumes that the predictors in each class have independent Gaussian distributions with the same variance (Hastie *et al.*, 2009:652).

Suppose there exists a test data case vector with expression levels $\boldsymbol{x}^* = \left(x_1^*, x_2^*, \dots, x_p^*\right)^T$, then the DLDA discriminant score for class g is

$$\delta_g(\boldsymbol{x}^*) = -\sum_{j=1}^{p} \frac{\left(x_j^* - \bar{x}_{jg}\right)^2}{s_j^2} + 2log\pi_g. \qquad (2.3)$$

Here s_j is the pooled within-class standard deviation for the j^{th} variable and $\bar{x}_{jg}$ is the sample mean for Variable *j* within class g. The first part of the equation in (2.3) is the negative standardised squared distance of $\boldsymbol{x}^*$ to the g^{th} centroid and the second part is the correction based on the class prior probability, π_g, where $\sum_{g=1}^{G} \pi_g = 1$. The classification rule of DLDA is then

$$C(\boldsymbol{x}^*) = \ell \text{ if } \delta_l(\boldsymbol{x}^*) = max_g \delta_g(\boldsymbol{x}^*).$$

The DLDA assumption is somewhat unrealistic, since predictors will rarely be independent within a class. Hastie *et al.* (2009:652) state that the assumption of independence applied in DLDA greatly reduces the number of parameters used in the model and often results in an effective and interpretable classifier. Furthermore, Bickel and Levina (2004) state that the DLDA classifier is often effective, especially in high-dimensional settings, and theoretically it will often outperform standard LDA in such problems. The main drawback of the DLDA classifier is that is uses all the predictors and therefore is not convenient for interpretation in high-dimensional problems (Hastie *et al.*, 2009:652). However, applying regularisation that automatically eliminates predictors that do not contribute to accurate classification using DLDA may lead to an improvement in terms of both interpretability and test error.

To summarise, the idea of class-wise independence in classification problems forms the basis for DLDA. Furthermore, if it is assumed that $X_1, \dots, X_p$ are independent within each class, and a LASSO type penalty is incorporated, the resulting classifier is the nearest shrunken centroid procedure proposed by Tibshirani *et al.* (2001) which is explained in detail in the next section.

2.5 NEAREST SHRUNKEN CENTROIDS

The nearest shrunken centroids (NSC) procedure was proposed by Tibshirani *et al.* (2001) and aims to identify a subset of predictor variables which best characterises the different classes (or the smallest subset of predictors which can accurately classify samples). NSC is a modification of the nearest centroid method and can be applied to classification problems as well as in unsupervised problems (Tibshirani *et al.,* 2001:6567). Nearest centroid classification applied to a test case $\boldsymbol{x}$ computes the (Euclidean) distance between $\boldsymbol{x}$ and each of the class centroids and then classifies $\boldsymbol{x}$ to the class whose centroid is closest to $\boldsymbol{x}$. The main drawback of nearest centroid classification is that is uses all the predictors, which is not desirable in for example microarray problems where $p \gg N$. NSC modifies this method by "shrinking" each of the class centroids towards the overall centroid for all the classes. The idea behind NSC is that predictors whose class specific centroids are close to the overall mean centroid should not play a role in classification. Therefore, NSC continuously shrinks the predictor variables until only a few have an influence on the classification. After shrinking the class centroids, a test sample is then classified by the usual nearest centroid rule, but using the shrunken, hopefully denoised class centroids. In NSC, one must choose a shrinkage parameter, Δ, which influences the number of predictors used in computing the shrunken centroids. If $\Delta = 0$, then all the predictors are used to compute the centroids. Tibshirani *et al.* (2001) propose that cross-validation should be used to determine the optimal value of Δ, denoted by Δ_{opt}.

This section proceeds with a more detailed description of NSC, based on Tibshirani *et al.* (2001) and Hastie *et al.* (2009, Section 18.2). Let x_{ij} denote the values for predictor j, $j = 1,2,\ldots,p$ and observations $i = 1,2,\ldots,N$. Also, let $\bar{x}_{jg}$ be the centroid (mean value) of variable j in class g. The overall centroid (mean value) for the j^{th} predictor is given by $\bar{x}_j = \sum_{i=1}^{N} \frac{x_{ij}}{N}$.

In the binary case, it is clear that if the group-specific mean values $\bar{x}_{j1}$ and $\bar{x}_{j2}$ do not differ significantly from the overall centroid $\bar{x}_j$ (and therefore $\bar{x}_{j1}$ and $\bar{x}_{j2}$ do not differ significantly from each other), then input variable j will most probably not play a significant role in accurate classification. The NSC method shrinks the class-specific, $\bar{x}_{jg}$, towards the overall centroids, $\bar{x}_j$. If the shrinkage takes $\bar{x}_{j1}$ and $\bar{x}_{j2}$ all the way to the overall centroid, variable X_j is effectively removed from the NSC classifier.

Consider the more detailed explanation for the general case of G groups. Let d_{jg} represent the shrunken difference for the j^{th} predictor. This is similar to a t-statistic for predictor j which compares the mean of class g to the overall centroid, namely

$$d_{jg} = \frac{\bar{x}_{jg} - \bar{x}_j}{m_g(s_j + s_0)}, \tag{2.4}$$

where $m_g = \sqrt{\frac{1}{N_g} + \frac{1}{N}}$ and s_j is the pooled within-class standard deviation for predictor j. Assuming $G = 2$,

$$s_j^2 = \frac{1}{N-2}\left(\sum_{i \in C_1}(x_{ij} - \bar{\boldsymbol{x}}_{j1})^2 + \sum_{i \in C_2}(x_{ij} - \bar{\boldsymbol{x}}_{j2})^2\right).$$

Therefore $m_g s_j$ is equal to the estimated standard error of the numerator in d_{jg} in (2.4).

In (2.4), s_0 denotes a small positive constant (regularisation parameter) which guards against large d_{jg} values that arise from predictors with near zero values. Tibshirani *et al.* (2001:6568) state that s_0 is typically set equal to the median of the s_j-values over the set of predictors, and has the same value for all the predictors. Tibshirani *et al.* (2003:107) explain that Equation (2.4) takes the size of each class into consideration and effectively applies a larger threshold to smaller (high variance) classes.

Equation (2.4) can also be written in the form

$$\bar{x}_{jg} = \bar{x}_j + m_g(s_j + s_0)d_{jg}.$$

Note that the normalisation by s_j in (2.4) has the effect of giving higher weight to predictors that have stable expression within observations of the same class (Tibshirani *et al.*, 2003:107).

The procedure now shrinks d_{jg} towards zero using the soft thresholding operator, giving

$$d'_{jg} = sign(d_{jg})(|d_{jg}| - \Delta)_+, \tag{2.5}$$

where Δ is determined by choosing the threshold that yields the minimum cross-validation misclassification error rate. In (2.5) each $|d_{jg}|$ is reduced by an amount Δ until it reaches zero. The subscript in (2.5) $(\,)_+$, indicates positive part, therefore $t_+ = t$ if $t > 0$, and zero otherwise. This is shown in Figure 2.2 below.

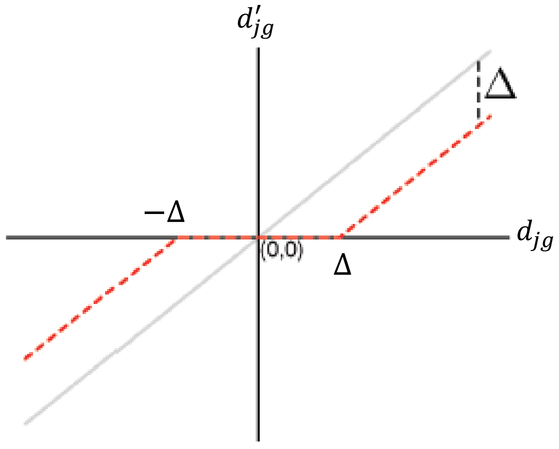

Figure 2.2: Soft thresholding function used in the NSC methodology

Source: Tibshirani *et al.,* 2003:106

In Figure 2.2, the soft thresholded function given in (2.5) is shown by the red dotted line, and the 45° line is shown in grey. The black bold dotted line between the red and grey lines represents Δ. If a predictor has $d'_{jg} = 0$ for all classes, then the centroid for predictor j, $\bar{\boldsymbol{x}}_j$ is the same for all classes and therefore the predictor does not contribute to classification. In Figure 2.2 it is clear that as Δ becomes larger, there will be more predictors with $d'_{jg} = 0$ and therefore more predictors are eliminated, leaving only the most significant predictors as discriminating features. It is clear that the tuning parameter in this process is the quantity Δ and that care has to be taken to specify its value. It is recommended that the value of Δ should be determined using cross-validation (CV). In such a process one typically chooses the largest value of Δ (resulting in the smallest number of predictors) that achieves the minimal cross-validation error (Tibshirani *et al.,* 2003:107). Note that in scenarios where there are several solutions with minimal cross-validation error rates, the optimal threshold value (number of genes) can either by chosen as the value with the largest likelihood or the one with the smallest number of genes (larger penalty). For the purpose of this thesis the latter method will be used if there is more than one threshold value yielding the minimal CV error rate.

Note that soft thresholding, as implemented in the NSC procedure, typically produces more reliable estimates of the true means since many of the $\bar{x}_{jg}$ values will be noisy and close to the overall mean $\bar{x}_j$ (Donoho and Johnstone, 1994).

Hard thresholding can be used as an alternative to the soft thresholding in (2.5) (Tibshirani *et al.,* 2003:111). Hard thresholding retains all the predictors with differences greater than Δ in absolute value and discards the other predictors. This implies that (2.5) changes to

$$d'_{jg} = d_{jg} \cdot I\left(\left|d_{jg}\right| > \Delta\right).$$

From the equation above it is clear that hard thresholding differs from soft thresholding shown in (2.5) in the sense that instead of shrinking all the d_{jg}'s by an amount Δ towards zero (as in (2.5)), the d_{jg}'s larger than Δ remain unchanged. The main disadvantage of using hard thresholding is that it has a "jumpy" nature: as Δ is increased a predictor that has a full contribution d_{jg} is suddenly set to zero (Tibshirani *et al.,* 2003:111). This typically leads to a procedure with larger variance than one based on soft thresholding.

Returning to the soft threshold NSC procedure, the new shrunken versions of $\bar{x}_{jg}$ are calculated by reversing the transformation in (2.4):

$$\bar{x}'_{jg} = \bar{x}_j + m_g\left(s_j + s_0\right)d'_{jg}.$$

These shrunken centroids $\bar{x}'_{jg}$ are substituted in place of $\bar{x}_{jg}$ in the discriminant score. This procedure therefore results in variable selection as only variables with a non-zero d'_{jg} for at least one of the classes play a role in the classification rule and typically a large number of variables can be discarded. Therefore, NSC actually implements variable selection and thereby dimension reduction.

As mentioned previously, the discriminant score defined for each data case in NSC is similar to that of diagonal LDA (Tibshirani *et al.,* 2001:6569). Suppose there exists a test data case vector with expression levels $\boldsymbol{x}^* = (x_1^*, x_2^*, \dots, x_p^*)$, then the discriminant score for class g is defined as

$$\delta_g(\boldsymbol{x}^*) = -\sum_{j=1}^{p} \frac{\left(x_j^* - \bar{x}_{jg}{'}\right)^2}{\left(s_j + s_0\right)^2} + 2log\pi_g. \qquad (2.6)$$

The first term in (2.6) standardises the squared distance of $\boldsymbol{x}^*$ to the g^{th} shrunken centroid. The second term, $2log\pi_g$, makes a correction based on the class prior probability π_g. Note that $\sum_{g=1}^{G} \pi_g = 1$ and that π_g provides the overall relative frequency of class g in the population. The prior probability π_g is typically estimated by the sample prior $\hat{\pi}_g = \frac{N_g}{N}$; however, if the sample prior is not representative of the population then more realistic or equal priors can be used, *i.e.* $\pi_g = \frac{1}{G}$ (Tibshirani *et al.,* 2001:6569). Tibshirani *et al.* (2001:6569) explain that “if the smallest distances are close and hence ambiguous, the prior correction in (2.6) gives a preference for larger classes, because they potentially account for more errors.”

The NSC classification rule is then

$$C(\boldsymbol{x}^*) = \ell \text{ if } \delta_l(\boldsymbol{x}^*) = max_g \delta_g(\boldsymbol{x}^*).$$

Estimates of the class probabilities, analogous to Gaussian linear discriminant analysis, can be constructed using the discriminant scores in (2.6):

$$\hat{p}_g(\boldsymbol{x}^*) = \frac{e^{\frac{1}{2}\delta_g(\boldsymbol{x}^*)}}{\sum_{l=1}^{G} e^{\frac{1}{2}\delta_l(\boldsymbol{x}^*)}}. \tag{2.7}$$

Hastie *et al.* (2009:654) point out several uses for these posterior probabilities.

Although the NSC method is explained above for two classes, it can readily be applied in scenarios with more than two classes. With $G > 2$, Tibshirani *et al.* (2001:6572) explain that NSC uses soft thresholds for all the differences between the class centroids and the overall centroids, and chooses a different set of predictors for characterising each class. The reader is referred to this paper for more detail in this regard.

The NSC procedure has two main advantages compared to nearest centroid classification, namely: it increases the accuracy of the classifier by reducing the effect of noise variables, and it performs automatic variable selection. Furthermore, the computations involved in the NSC method are straightforward and the authors in Tibshirani *et al.* (2001) have created an *R* package called “Prediction Analysis for Microarrays”, *pamr*, which implements the NSC methodology.

Finally, Tibshirani *et al.* (2003:112) state that the NSC procedure will fail in cases where the centroids for all the classes are the same, implying $\bar{x}_{jg} = \bar{x}_j$ and therefore in (2.4), $d_{jg} = 0$. In such a scenario, there will be no shrinkage and all the predictors play a role in the classification rule. This holds true in scenarios for most classifiers that make use of the class centroids, for example LDA.

The NSC classifier can be implemented in high-dimensional problems and will be applied later in the thesis as a classifier in both the binary and multi-class data sets considered.

Furthermore, NSC can be applied as a variable selection or dimension reduction tool per se for high-dimensional data sets. In this regard, it is possible to apply NSC thresholding to implement variable selection, followed by application of a classifier such as KNN or an SVM based only on the selected variables. This therefore entails using the $\left(|d_{jg}| - \Delta\right)_+$ part of Equation (2.5) as a variable selecyion method, by selecting only the variables with a non-zero value of $\left(|d_{jg}| - \Delta\right)_+$. The remaining variables are discarded and do not play a role in the final classification rule. The optimal value for the threshold is determined using Leave-one-out CV (LOOCV) on the training set, where the optimal value for the threshold is the highest threshold candidate value (smallest number of selected variables) which achieves the lowest LOOCV error rate.

A detailed explanation of how the NSC procedure is applied - either as a classifier or as a variable selection technique in this thesis - is provided in Chapter 5. Details regarding the necessary software are also provided in that chapter.

2.5 SUPPORT VECTOR MACHINES

Support vector machines are popular classifiers due to their sound theoretical background and good performance in practical applications. The basic theory and mathematics behind the concept of SVMs was developed by Vapnik and Lerner (1963) and Vapnik and Chervonenkis (1964). However, the present form of the SVM was formalised and introduced by Böser *et al.* (1992). The explanation of SVMs in this section is based on the discussion in Hastie *et al.* (2009, Chapters 4 and 12).

SVMs can be used for classification for linearly separable and linearly non-separable datasets. The case of linearly separable training data is considered first. SVMs for linearly separable datasets are called linear SVMs or optimal separating hyperplanes. If a binary classification dataset is linearly separable then there exists a hyperplane in the p-dimensional input space that perfectly separates the data cases from the two populations. A hyperplane is defined as the set in $\Re^p$ satisfying a linear restriction of the form

$$\{\boldsymbol{x}: f(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^T\boldsymbol{x} = 0\}. \tag{2.8}$$

A hyperplane in $\Re^p$ is therefore a flat affine subspace of dimension $(p-1)$, where affine indicates that the subspace does not need to pass through the origin (James *et al.*, 2013:338). Assuming that the response variable is coded as $Y = -1$ or $Y = 1$ to denote the two classes, the classifier corresponding to $f(\boldsymbol{x})$ in Equation (2.8) is $G(\boldsymbol{x}) = sign[f(\boldsymbol{x})]$.

The linear SVM is the maximum margin hyperplane, where the *margin* of a hyperplane is defined to be the maximum width of a slab that can be placed around the hyperplane without reaching any of the data points. In order to formalise this definition in terms of an optimisation problem, note that if the two populations are linearly separable, there exists a function $f(\boldsymbol{x}) = \beta_0 + \boldsymbol{\beta}^T\boldsymbol{x}$ satisfying $y_i f(x_i)$ >0 $\forall i$, *i.e.* there is correct classification for each training case. Note once again that the assumption of $y_i = \pm 1$ is made here, and throughout the discussion of SVMs. Finding the maximum margin hyperplane entails solving the following optimisation problem:

$$\underset{\boldsymbol{\beta},\beta_0,||\boldsymbol{\beta}||=1}{\text{maximise}}\, M$$

$$\text{subject to } y_i(\beta_0 + \langle\boldsymbol{\beta},\boldsymbol{x}_i\rangle) \geq M, \quad i = 1,\dots,N.$$

Here M represents the margin of the hyperplane. Note that there are in fact two interpretations of the term “margin”: the first interpretation refers to the margin of a linearly separable binary classification data set, and the second interpretation is the margin value for an individual data point, defined to be $y(\beta_0 + \boldsymbol{\beta}^T\boldsymbol{x})$.

The set of conditions in the optimisation problem aim to ensure that all the points are on the correct side of the hyperplane (provided M is positive) and at least a signed distance M from the decision boundary defined by $\boldsymbol{\beta}$ and β_0. Hastie *et al.* (2009:132) show how the restriction $||\boldsymbol{\beta}|| = 1$ can be eliminated, and how, by setting $||\boldsymbol{\beta}|| = \frac{1}{M}$, the following convex optimisation problem can be obtained:

$$\underset{\boldsymbol{\beta},\beta_0}{\text{minimise}}\left(\frac{1}{2}||\boldsymbol{\beta}||^2\right)$$

$$\text{subject to } y_i(\beta_0 + \langle\boldsymbol{\beta},\boldsymbol{x}_i\rangle) \geq 1, \quad i = 1,\dots,N. \qquad (2.9)$$

The following description of an efficient procedure to solve this optimisation problem is based on the discussion in Hastie *et al.* (2009:132-133). Introduce non-negative Lagrange multipliers $\alpha_1, \alpha_2, \dots, \alpha_N$, a powerful tool for solving constrained optimization problems of differentiable functions. The Lagrange dual function obtained in this case has the form

$$\max L_D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\, \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle, \tag{2.10}$$

where $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ denotes the usual inner product between two vectors. This quantity has to be maximised with respect to $\alpha_1, \alpha_2, \dots, \alpha_N$ subject to

$$\alpha_i \geq 0 \text{ and} \sum_{i=1}^{N} \alpha_i y_i = 0.$$

Note that the expression in Equation (2.10) for the dual Lagrangian is now in quadratic form, while the constraints are all linear in the $\alpha_i's$. Therefore, this is a quadratic programming problem which can easily be solved using standard optimisation techniques, thereby obtaining the optimal values of $\alpha_1, \alpha_2, \dots, \alpha_N$.

Quadratic programming theory implies that the so-called Karush-Kuhn-Tucker conditions have to be satisfied at the solution of the optimisation problem. One of these conditions provides an expression for the optimal slope vector, *viz.*

$$\boldsymbol{\beta} = \sum_{i=1}^{N} \alpha_i y_i\, \boldsymbol{x}_i.$$

Another of these conditions leads to the important sparsity property of the SVM, *i.e.* the fact that the final SVM classifier depends only on a subset of the training data. The relevant condition in this regard is $\alpha_i[y_i(\beta_0 + \boldsymbol{\beta}^T\boldsymbol{x}_i) - 1] = 0$ for all values of $i = 1,2,\dots,N$. It follows from this condition that if $\alpha_i > 0$, then $y_i(\beta_0 + \boldsymbol{\beta}^T\boldsymbol{x}_i) = 1$, implying that the sample point $\boldsymbol{x}_i$ lies on the boundary of the separating hyperplane slab. Similarly, if $y_i(\beta_0 + \boldsymbol{\beta}^T\boldsymbol{x}_i) > 1$, and therefore $\boldsymbol{x}_i$ lies outside the slab, then $\alpha_i = 0$. It follows that the summation in the above expression for the optimal $\boldsymbol{\beta}$ can be limited to a summation over only the so-called *support vectors*, *i.e.* the data points having $\alpha_i > 0$.

Having found the optimal values of $\alpha_1, \alpha_2, \dots, \alpha_N$, and thereby also the optimal value of $\boldsymbol{\beta}$, the only remaining problem is to determine a value for the intercept, β_0. This is done by solving for β_0 using $y_i(\beta_0 + \boldsymbol{\beta}^T\boldsymbol{x}_i) = 1$ for all the support vector cases, and computing the average of these solutions. The resulting *optimal separating hyperplane classifier* is then given by

$$\hat{f}(\boldsymbol{x}) = sign(\beta_0 + \boldsymbol{\beta}^T\boldsymbol{x}),$$

which is equivalent to

$$\hat{f}(\boldsymbol{x}) = sign\left(\beta_0 + \sum_{i=1}^{N} \alpha_i y_i \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle\right).$$

Note that the summation appearing in the last expression above will only be over all the support points, which will often be a relatively small number.

The thesis now turns to a discussion of the much more frequently occurring case of training data that are not linearly separable. The optimal separating hyperplane described above is first extended to the *support vector classifier*, which can deal with overlapping classes. The *support vector machine* is then obtained as a technique which fits a linear classifier in a transformed version of the input space. The important role played by the kernel trick in the support vector machine is emphasised.

One option to handle linearly non-separable data is to modify the criterion to be optimised by introducing an additional cost associated with the misclassification of training data cases. The additional cost associated with misclassification is introduced by relaxing the constraints leading to the optimal separating hyperplane by introducing so-called (non-negative) slack variables, $\xi_i, i = 1, \dots, N$. This approach leads to the optimisation problem in (2.9) becoming

$$\min_{\boldsymbol{\beta_0}, \boldsymbol{\beta}} \left\{\frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\sum_{i=1}^{N} \xi_i\right\},$$

subject to $\xi_i \geq 0$ and $y_i(\beta_0 + \boldsymbol{\beta}^{\boldsymbol{T}}\boldsymbol{x}_i) \geq 1 - \xi_i, \quad i = 1, \dots, N.$

Here the tuning or regularisation parameter C of the support vector classifier denotes the cost associated with an observation falling on the wrong side of the decision boundary. In practical applications, the value of C is usually determined by means of cross-validation. Note that a large value of C heavily penalises a positive slack variable, thereby leading to a fairly narrow slab for the eventual decision boundary. A similar interpretation holds for small values of C.

The technical theory leading to a solution of the optimisation problem for the support vector classifier is similar to that for the optimal separating hyperplane and will not be discussed in detail here. The interested reader is referred to Hastie *et al.* (2009:419-421) for details. The final classifier is once again of the form $\hat{f}(\boldsymbol{x}) = sign\left(\beta_0 + \sum_{i=1}^{N} \alpha_i y_i \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle\right)$, with the intercept parameter determined as before.

This section turns finally to a discussion of the support vector machine. The support vector classifier results in a linear decision boundary in the original input space. This is not always sufficiently flexible. The support vector machine is based on the idea of transforming the input vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N$ to feature vectors $\phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2), \dots, \phi(\boldsymbol{x}_N)$ and to apply the support vector classifier algorithm in the transformed space. This results in a linear decision boundary in feature space which, however, corresponds to a non-linear boundary in input space. Interestingly, the feature transformation often leads to an infinite-dimensional feature space. The question then arises as to how the required computations for fitting and using a classifier can be performed in an infinite-dimensional space. An elegant answer to this question is based on the fundamental fact that the input patterns appear in the support vector classifier only in terms of inner products. This makes it possible to use the important *kernel trick* to perform computations in feature space. More specifically, an inner product of the form $\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$ can be evaluated in terms of a kernel function, $K(.,.)$. A kernel function is a symmetric positive-definite function defined on $\Re^p \times \Re^p$ and the kernel trick states that $\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. This result obviates the need to do explicit calculations in a (possibly infinite-dimensional) feature space.

Summarising, the rationale behind using an enlarged input space is that generally linear boundaries in the enlarged space achieve better training-class separation (Hastie *et al.*, 2009:423). This principle is illustrated in Figure 2.3 below.

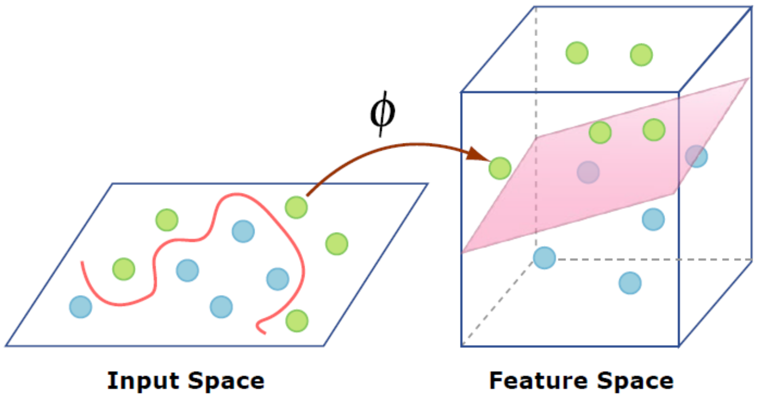

Figure 2.3: A graphical representation summarising the kernel trick in SVM

Source: Markowetz, 2005

The role of the cost or regularisation parameter, C, is clearer in the enlarged feature space since perfect separation is then often achievable (Hastie *et al.,* 2009:424). A large C will discourage any positive slack variables, ξ_i, and may possibly lead to an overfit wiggly boundary in the original optimal space to ensure that none of the points lie on the wrong side of the decision boundary. Conversely, a small value of C will encourage a small value of $||\boldsymbol{\beta}||$ and consequently $f(\boldsymbol{x})$ and the decision boundary will be smooth and possible lead to underfitting. Therefore, the value of C should be determined using cross-validation.

This thesis briefly indicates how the above ideas can be implemented in order to arrive at the SVM classifier. Replacing $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ in the earlier discussion by $\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$, the Lagrange dual function is found to be

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \, \alpha_j y_i y_j \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle, \tag{2.11}$$

which involves $\phi(\boldsymbol{x})$ only through inner products. The result $\boldsymbol{\beta} = \sum_{i=1}^{N} \alpha_i \, y_i \boldsymbol{x}_i$ now becomes $\boldsymbol{\beta} = \sum_{i=1}^{N} \alpha_i y_i \phi(\boldsymbol{x}_i)$ and the solution function $f(\boldsymbol{x})$ can be written as $f(\boldsymbol{x}) = \beta_0 + \phi(\boldsymbol{x})^T \boldsymbol{\beta}$**,** which implies that

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \, y_i \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}_i) \rangle + \beta_0. \tag{2.12}$$

It is clear once again that only inner products between feature vectors need to be computed in order to evaluate (2.12). The intercept, β_0, can once again be determined as before, and it transpires that this also involves the feature vectors only in terms of inner products.

At this point the kernel trick can be used to compute the inner products appearing in the above expressions. As pointed out above, the kernel trick is the key to solving the computational problems by using a kernel function to compute the inner products in the enlarged transformed space, *viz.*

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle = K(\boldsymbol{x}, \boldsymbol{x}').$$

Hastie *et al.* (2009:424) state that the kernel function K should be a symmetric positive (semi-) definite function and that the choice of kernel function is crucial to ensure that the resulting SVM will generalise well. These authors provide three popular choices for the kernel function in the SVM literature, *viz.*

1. dth- degree polynomial: $K(\boldsymbol{x},\boldsymbol{x}') = (1 + \langle \boldsymbol{x},\boldsymbol{x}' \rangle)^d$, where d is an integer value
2. Radial Basis: $K(\boldsymbol{x},\boldsymbol{x}') = \exp(-\gamma\|\boldsymbol{x}-\boldsymbol{x}'\|^2)$
3. Neural Network: $K(\boldsymbol{x},\boldsymbol{x}') = \tanh(1 + \langle \boldsymbol{x},\boldsymbol{x}' \rangle)$.

Hastie *et al.* (2009:659) state that radial basis function (RBF) kernels tend to dampen the inner products between points far away from each other, which in turn leads to robustness to outliers. This often occurs in high dimensions, and therefore RBF kernels often deliver superior results over other kernels in high-dimensional data sets. Therefore, this thesis will focus on the (Gaussian) RBF kernel, which has the form $K(\boldsymbol{x},\boldsymbol{x}') = \exp(-\gamma\|\boldsymbol{x}-\boldsymbol{x}'\|^2)$. Here γ is a positive real number representing the scale parameter. Empirical evidence suggests that when fitting an SVM with a Gaussian radial kernel, $\gamma = \frac{1}{p}$ generally works well. In the RBF kernel, γ is a tuning parameter; a large γ results in more wiggly decision boundaries (as the fit becomes more non-linear) whilst for a small γ the decision boundaries get smoother. An SVM with RBF kernel has very local behaviour, as only nearby training observations, in terms of Euclidean distance, have an effect on the classification of a test observation (James *et al.*, 2013:353).

Summarising, since both (2.11) and (2.12) involve $\phi(\boldsymbol{x})$ only through inner products, the transformation of $\phi(\boldsymbol{x})$ does not need to be specified and from (2.12) the SVM classifier can be written as

$$\hat{f}(\boldsymbol{x}) = \hat{\beta}_0 + \sum_{i=1}^{N} \hat{\alpha}_i y_i K(\boldsymbol{x},\boldsymbol{x}_i).$$

There are N parameters, $\hat{\alpha}_i$, that need to be estimated (one for each training observation), but as pointed out before, $\hat{\alpha}_i > 0$ only for the support vectors. Therefore, the SVM only depends on a fraction of the training points, referred to as support vectors.

The SVM function $\hat{f}(\boldsymbol{x})$ will produce a real value, and $\hat{G}(\boldsymbol{x}) = sign[\hat{f}(\boldsymbol{x})]$ needs to be calculated in order to classify $\boldsymbol{x}$. If $\hat{G}(\boldsymbol{x}) > 0$, $\boldsymbol{x}$ is classified to the $+1$ class and conversely, if $\hat{G}(\boldsymbol{x}) < 0$, $\boldsymbol{x}$ is classified to the -1 class. Note that if it is a more serious mistake to classify a -1 case to the $+1$ class than the other way around, then a possible approach could be to change the value $\hat{\beta}_0$ to achieve a lower (training) error rate for one type of misclassification by paying the price of a higher training error for the other type of misclassification.

It is important to note that the kernel property is not unique to SVMs. It turns out that many other standard statistical procedures also depend on the training inputs only in terms of inner products. An important example is linear discriminant analysis. Any such technique can be kernelised, entailing simply replacing ordinary inner products by kernel function evaluations. If this is done in the case of linear discriminant analysis, a technique known as Kernel Fisher Discriminant Analysis (KFDA) is obtained. Such kernelised techniques, including SVMs, are generally considered to be highly flexible models with however low interpretability.

In Chapter 5 the SVM classifier will be applied to the practical data sets considered there due to its powerful ability to accommodate non-linear class boundaries by using the kernel trick. However, a disadvantage of using SVMs is its difficult application which requires the correct choice of several tuning parameters. It turns out that it is crucial to perform an extensive search using different tuning parameter combinations to achieve reliable results.

2.6 SUMMARY

This chapter contains a description of the base classifiers which will be used in the empirical study reported in Chapters 5 and 6. The classifiers considered are: *linear discriminant analysis*, a technique which has been successfully applied in classification problems over many years; K*-nearest neighbours*, a very simple non-parametric technique which is frequently successfully applied in low-dimensional problems; *diagonal linear discriminant analysis*, a modification of the traditional version based on a simplifying independence assumption and specifically designed for very high-dimensional problems; *nearest shrunken centroids*, which combines the simplifying assumption underlying diagonal linear discriminant analysis with a simple and effective variable elimination strategy, and finally, the *support vector machine*, a recent addition to the statistician's toolbox, originating in the field of machine learning and showing excellent performance in many applications.

The next chapter is devoted to the topic of variable selection, an important strategy in high-dimensional scenarios.

CHAPTER 3
VARIABLE SELECTION

3.1 INTRODUCTION

The identification of significant variables is an essential preliminary step in dimension reduction when a data set contains potentially important variables together with a large number of irrelevant or noise variables. Although many variable selection (VS) techniques were originally developed for regression, many of these techniques can be applied in classification as well. Variable selection attempts to determine the strength of the relationship between the predictor or classification variables and the response variable. In classification, the stronger the relationship between a classification variable and the response variable, the greater the difference between the values of the classification variable for the different populations corresponding to the different values of the response variable. Predictor variables that have a weak relation with the response variable, *i.e.* those which do not clearly distinguish between the different populations, should not be selected for inclusion into the model. Therefore, variable selection serves a dual purpose in statistical classification problems: it enables one to identify the input variables which separate groups well, and a classification rule based on these variables frequently has a lower error rate than the rule based on all the input variables.

One of the main challenges faced in model selection when $p \gg N$ is the presence of collinearity among the predictor variables which can lead to the problem of overfitting and model misidentification (Fan and Lv, 2010:102). Consequently, variable selection is essential for addressing the issue of collinearity in a high-dimensional setting. The paper by Fan and Fan (2008) explored the impact of dimensionality on classification and suggested that, due to the noise accumulation that exists in a high-dimensional setting, classification using all the predictors may perform as poorly as random guessing, highlighting the importance of variable selection when $p \gg N$.

In high-dimensional problems, traditional variable selection techniques, such as best subset selection methods, are computationally too expensive and consequently other forms of selection have been developed to overcome this problem (Fan and Lv, 2010). This chapter will discuss parts of the paper by Fan and Lv (2010) and other variable selection methods developed to overcome the curse of dimensionality. This will be followed by a discussion of pre-conditioning for feature selection as methods of performing variable selection in a high-dimensional setting.

There are two main reasons for applying variable selection to a data set: firstly, it makes interpretation easier, especially when p is large, and secondly, variable selection can lead to models yielding more accurate predictions or classifications than those based on all of the available variables.

Variable selection is especially relevant in modern applications, where p can frequently be very large. The objective of variable selection is to identify a subset of $\{X_1, \dots, X_p\}$ containing the inputs that are relevant or important when one wishes to explain variation in the response. Conceptually there are two parts to the variable selection problem. The first part requires finding the best (in some sense) subset of variables for every given number of inputs $r \leq p$. The second part entails deciding on the value of r, *i.e.* the number of inputs to include in the final model. The second part is usually the more difficult one, since it entails weighing up the benefit of a larger model that fits the training data better than a smaller model against the disadvantage of a more complex model that may tend towards overfitting.

There is a very large literature on the topic of variable selection. A comprehensive discussion of this literature will not be attempted. Instead, the thesis proceeds with a brief description of traditional variable selection approaches (all possible subsets regression, forward stepwise selection and backward stepwise selection), followed by comments on several more recently proposed approaches.

Hastie *et al.* (2009:57) define best-subset selection as a strategy which first of all finds the subset of given size r with the smallest residual sum of squares (RSS), the best subset of size r, for every given dimension. Regarding the optimal choice for r, the bias-variance trade-off combines with the principal of parsimony (Hastie *et al.*, 2009:57). Hence, in the first stage of a best-subset selection, for $r = 1,2,..,p$ all the $\binom{p}{r}$ possible models that contain exactly r predictors are fit to the data, and then the RSS or R^2 is calculated for each of these $\binom{p}{r}$ models. The best among these $\binom{p}{r}$ models is the model with the minimum RSS or equivalently the largest R^2. The second stage of the process is to select the single best model from the $p + 1$ possible models identified in stage one, therefore selecting the value of r. The most common criteria in choosing r are using cross-validation to determine the prediction error and other estimates for model error such as AIC, BIC, C_p or adjusted R^2. Despite best-subset selection being simple and conceptually appealing, it faces statistical problems when p is large and is only computationally feasible when $p < 40$ (James *et al.*, 2013:207).

Stepwise selection procedures consider a more restricted set of models and are attractive alternatives to best subset selection (James *et al.*, 2013:207). Forward stepwise selection starts with a model containing only the intercept, and then sequentially adds predictors into the model that most improve the fit (Hastie *et al.*, 2009:58). The forward stepwise selection algorithm is referred to as a greedy algorithm as it aims to do as well as possible at each step of the iterative procedure. The forward stepwise selection procedure starts with a null model ($\mathcal{M}_0$) containing no predictors (hence, only the intercept). The first step is to add the predictor which provides the largest additional improvement in fit to the null model, by determining the predictor with the smallest RSS or highest R^2. Adding this predictor gives $\mathcal{M}_1$. To get $\mathcal{M}_2$, consider all of the $p-1$ predictors which have not yet been entered into the model and once again determine the best model if one of these variables is added to $\mathcal{M}_1$. The process continues for $r = 0, \dots, p-1$, adding one predictor at a time to the previous model until all the possible predictors are in the model. The difference between best subset selection and forward selection therefore lies in the fact that in forward selection the procedure is not considering every possible model containing r predictors, but instead just adds one predictor to the previously identified model. The final step is to select a single best model from amongst the $p+1$ possible models $(\mathcal{M}_0, \dots, \mathcal{M}_p)$ using CV prediction error, C_p, AIC, BIC or adjusted R^2 (as these candidate models all have different sizes). The models $\mathcal{M}_0, \dots, \mathcal{M}_p$ are nested models implying that each model contains all the predictors in the previous model plus one additional predictor.

Hastie *et al.* (2009:58) describe the algorithm for backward stepwise selection as starting with the full model (all the predictors in the model), and then iteratively, one-at-a-time, deleting those predictors from the model that have the least impact on the fit. James *et al.* (2013:209) highlights that forward and backward stepwise selection both have a computational advantage over best subset selection as they consider approximately $1 + \frac{p(p+1)}{2}$ models, which is significantly less than the 2^p models in best subset selection

Under what circumstances are these methods no longer sufficient? The large value of p frequently occurring in modern data sets is the main reason why traditional variable selection methods are no longer sufficient. In the next sections, a discussion of several more recently developed approaches for variable selection is given.

3.2 THE LASSO, LAR AND FORWARD STAGEWISE SELECTION

In high-dimensional problems, traditional variable selection methods are no longer sufficient, mainly due to the large value of p. Therefore, modern procedures such as the least absolute shrinkage and selection operator (Tibshirani, 1996), forward stagewise regression (Hastie *et al.,* 2001) and least angle regression (Efron *et al.*, 2004) were developed to improve stability and predictions (Hesterberg *et al.,* 2008).

This section briefly describes three modern model-building algorithms, namely: the least absolute shrinkage and selection operator (LASSO), least angle regression (LAR) and forward stagewise selection. These three procedures all employ similar strategies, but differ according to the type of direction and the active set of variables used.

3.2.1 THE LASSO

The LASSO is an example of a *shrinkage* procedure (ridge regression is another familiar example). In such a procedure, a least squares model is fit to all p available predictors, however the estimated (least squares) coefficients are shrunk towards zero. Depending on the type of shrinkage performed, some of the estimated least squares coefficients may be shrunk to exactly zero, thereby effectively performing variable selection.

The LASSO is a more recent shrinkage method than ridge regression and it performs continuous shrinkage of the regression coefficients with eventual variable selection. The LASSO uses an ℓ_1 penalty which simultaneously performs shrinkage towards zero as well as variable selection. The latter LASSO characteristic stands in contrast to ridge regression, which always includes all p predictors in the final model. The LASSO coefficient estimates are defined by the following equation:

$$\hat{\boldsymbol{\beta}}^{lasso} = \underset{\boldsymbol{\beta}}{argmin} \left\{ \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}.$$

Here $\lambda \geq 0$ is a complexity parameter which penalizes the size of the coefficients and thus controls the amount of coefficient shrinkage. The ℓ_1 LASSO penalty is $\ell_1 = \sum_{j=1}^{p} |\beta_j|$ and including this term into the criterion that has to be optimised causes the coefficient estimates to be shrunk towards zero. Note that the ℓ_1 penalty is not applied to the intercept in the model. In the LASSO the ℓ_1 penalty has the effect of shrinking some of the coefficient estimates to be exactly equal to zero when the complexity parameter is sufficiently large. The shrinkage performed by the LASSO is called "soft thresholding". The LASSO is similar to best-subset selection in the sense that it performs variable selection. However, best-subset selection drops all predictor variables with absolute coefficients smaller than the M^{th} largest; this is a form of "hard thresholding" (Hastie *et al.*, 2009:69). A potential disadvantage of the LASSO in high-dimensional problems is that the number of variables retained in the LASSO model is bounded by $min\{p, N-1\}$. So, if p is much larger than N it may happen that the LASSO selects too few variables.

The soft thresholding that forms the basis of the LASSO has links to NSC which was discussed in Section 2.5.

3.2.2 LAR

After its introduction in 1996, the LASSO did not immediately become popular. This was mainly because its computation involved a time-consuming non-linear optimisation approach. Least angle regression, introduced in 2004 by Efron *et al.*, provided an extremely efficient algorithm for computing the entire LASSO solution (hence, for all possible values of the complexity parameter) in regression problems. LAR can be viewed as a "more democratic" (less greedy) version of forward stepwise regression (Efron *et al.*, 2004). It uses a strategy similar to that of forward stepwise regression, which is explained above, but only enters "as much" of a predictor as it deserves which explains why it is viewed as less greedy.

The LAR algorithm consists of five steps and is explained by Hastie *et al.* (2009:73-74) as follows: The first step in the LAR algorithm is to standardise the predictor variables to have mean zero and unit norm. Start with the current residual, $r = y - \bar{y},\ \beta_1 = \beta_2 = \cdots = \beta_p$ =0. The next step is to identify the predictor $X_j,\ j = 1, \ldots, p$ most correlated with r, and hence the current response. Instead of fitting X_j completely, LAR moves β_j continuously from zero towards its least squares value causing its correlation with the evolving residual to decrease in absolute value. As soon as another variable, say X_h, "catches up" in terms of correlation with the current residual, the process is paused. Therefore, Step 3 in the LAR algorithm is to increase β_j from 0 in the direction of its least-squares coefficient $\langle X_j, r\rangle$, until some other competitor X_h , $h \neq j$ has as much correlation with the current residual as does X_j. In Step 4, the second variable X_h joins the active set and $\hat{\beta}_j$ and $\hat{\beta}_h$ are moved together in the direction defined by their joint least squares coefficients of the current residual on (X_j, X_h). This is done until some other competitor X_l has as much correlation with the current residual. This process is continued until all p predictors have entered the model. At this point $corr(r, x_j) = 0\ \forall j$ and the full least squares solution has been reached.

Note that if $p > N - 1$, then after $\min(N - 1, p) = N - 1$ steps the LAR algorithm reaches a zero-residual solution and arrives at the full least-squares solution.

The paper by Hesterberg *et al.* (2008) describes three remarkable properties of LAR. The first property is the computational efficiency of the LAR algorithm: Efron *et al.* (2004) state that the entire sequence of LAR steps with $p < N$ variables requires the same order of computations as an ordinary least squares fit on p variables.

The second property is that a simple modification of the basic LAR algorithm can be used as an efficient and relatively simple way to fit the LASSO and the stagewise model, with certain modifications in higher dimensions (Efron *et al.*, 2004).

The third and final remarkable property of the LAR algorithm is the availability of a simple C_p statistic for choosing the number of steps in the LAR algorithm, namely

$$C_p = \frac{1}{\hat{\sigma}^2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 - N + 2M, \tag{3.1}$$

where M is the number of steps and $\hat{\sigma}^2$ is the estimated residual variance. This property is based on Theorem 3 in Efron *et al.* (2004:424) which states that after M steps in the LAR algorithm, the degrees of freedom of a LAR estimate is given by

$$\sum_{i=1}^{n}\frac{\mathrm{cov}(\hat{\mu}, Y_i)}{\sigma^2},$$

which is approximately equal to M. This provides a simple stopping rule for LAR, namely to stop after the number of steps M that minimises the C_p statistic in (3.1).

3.2.3 Forward-stagewise Selection

Forward-stagewise selection is a variable selection technique that is more constrained than forward stepwise regression (Hastie *et al.*, 2009:60). Efron *et al.* (2004) provide a detailed description of the algorithm for forward-stagewise regression. The forward-stagewise selection algorithm starts, like forward stepwise selection, with just the intercept in the model. The first step is once again to determine the correlation between each of the predictors and the response. Then the predictor with the largest absolute correlation is selected. Suppose this is variable X_j. Simple linear regression of y on X_j is performed to compute a residual (which is now considered the response). The coefficient from this regression is added to the current coefficient in the model of the chosen variable X_j. The process is repeated, identifying at each step the variable with the largest absolute correlation with the current residual, r. At each step the simple linear regression coefficient of the selected variables is computed and this value is then added to the current coefficient for that variable, which may or may not be zero (Hastie *et al.,* 2009:60). The process is repeated until none of the variables are correlated with the residuals, at which point the full least squares solution has been reached. Forward-stagewise selection is referred to as "slow fitting" as it requires more than p steps to achieve the least squares fit, making it inefficient and problematic in high-dimensional cases (Hastie *et al.*, 2009:60).

3.3 SURE INDEPENDENCE SCREENING

Sure independence screening (SIS) was first introduced by Fan and Lv (2008) to reduce computations in variable selection when there is a large number of input variables, $p \gg N$. SIS possesses the property that the independence screening technique retains all of the important variables in the model with probability tending to one. This property dramatically narrows down the search for important variables.

This approach proceeds by calculating, for each input variable, a variable importance criterion, $w(X, Y)$, which measures the dependence between variables X and Y. The basic idea behind SIS is to compute the variance importance criterion

$$w_j = w(X_j, Y), \qquad j = 1, \dots, p \tag{3.2}$$

and then rank these values decreasingly: $w_{(1)} \geq w_{(2)} \geq \cdots \geq w_{(p)}$. The next step is to determine a threshold, possibly from the data, denoted by Δ and then retain/select all the variables having $w_{(j)} \geq \Delta$.

There are two questions that arise in the SIS procedure: Firstly, what measure, $w(\cdot)$, should be used? And secondly, how to determine a value for the threshold parameter Δ?

Applications of this technique later in the thesis are in the context of classification problems. Hence, focus is now on the case where $Y \in \{0,1\}$, *i.e.* the binary classification scenario. In this thesis three examples of variable importance criteria for binary classification problems will be considered:

i. The correlation coefficient

ii. The p-value of the two-sample t-test

iii. The score statistic.

These three examples will be discussed briefly in the following sections.

Although there are several thresholding strategies, cross-validation on possible candidate values is the most prominent method for determining a value of Δ and is the method preferred in this thesis. The possible candidate values can either be pre-specified or obtained from the data depending on the variable importance criterion. The method of obtaining the candidate values will be discussed for each variable importance criterion.

Another popular method of independence screening is the false discovery rate (FDR) proposed by Benjamini and Hochberg (1995) which focuses on controlling the proportion of genes falsely identified as significant. The FDR is commonly used in multiple testing as a method for controlling the expected false positive rate. In the microarray setting, the FDR is the expected proportion of false positive genes among the list of genes that are known to be significant.

3.3.1 Correlation Coefficient

An example of SIS is ranking the predictors according to the magnitude of their absolute sample correlation coefficients with the response variable. Therefore, using the correlation coefficient as the variable importance criterion for a binary classification problem entails computing

$$w_j = abs\left(corr(X_j, Y)\right),\ j = 1, \dots, p.$$

These values are then ranked decreasingly, $w_{(1)} \geq w_{(2)} \geq \cdots \geq w_{(p)}$, and the data set is reduced to the best specified percentage with the highest correlation or to those predictors whose absolute correlations exceed a certain threshold, $w_j \geq \Delta$. The latter method is implemented as a method of VS in this thesis. The correlation coefficient threshold that should be used in VS is dependent on the data set in question, therefore in this thesis a range of five possible correlation threshold values are considered for each data set. This process will be explained in detail in the empirical study in Chapter 5.

3.3.2 Two-sample t-test thresholding

Two-sample t-tests are a classic variable selection procedure, which ranks the predictor variables according to their ability and importance in distinguishing between two population groups (Lu and Petkova, 2014:403). The procedure determines the difference between the two population groups for each variable, and eliminates the variables with the least significant differences. Two-sample t-tests is a commonly used VS procedure for binary classification. However, in classification of high-dimensional data two-sample t-tests are usually applied as a pre-screener (*e.g.* in the same manner as SIS) rather than a variable selection method (Lu and Petkova, 2014:403).

The first step in application of this approach is to calculate the two-sample t-statistics for all the variables as follows,

$$t_j = \frac{\bar{\boldsymbol{x}}_{2j} - \bar{\boldsymbol{x}}_{1j}}{\mathrm{se}_j}, \tag{3.3}$$

where $\bar{\boldsymbol{x}}_{2j}$ and $\bar{\boldsymbol{x}}_{1j}$ represent the mean value of variable j for the cases in Group 2 and in Group 1 respectively. Also, se_j in (3.3) represents the usual pooled within-group standard error for variable j, given by

$$\mathrm{se}_j = \hat{\sigma}_j \sqrt{\frac{1}{N_1} + \frac{1}{N_2}}\ , \ \ j = 1, 2,$$

where N_1 and N_2 denote the number of observations in Group 1 and Group 2 respectively and where

$$\hat{\sigma}_j^2 = \frac{1}{N_1 + N_2 - 2}\left(\sum_{i \in C_1} \left(x_{ij} - \bar{\boldsymbol{x}}_{1j}\right)^2 + \sum_{i \in C_2} \left(x_{ij} - \bar{\boldsymbol{x}}_{2j}\right)^2\right).$$

Here C_1 and C_2 are the sets of indices of the observations in Group 1 and Group 2 respectively. Note that this form of the two-sample t-test statistics is based on the assumption that the two population variances are equal. Appropriate modifications can be made if this assumption is not valid, but the effect on the eventual variable screening will probably be only slight.

After the t-test statistics have been computed, the variables are sorted in descending order according to the absolute values of their t-statistics. Equivalently, sorting can be done based on the p-values corresponding to the t-test statistics. A thresholding rule is then used to select the top-ranking variables which will be used as the relevant variables. There are several thresholding rules available for binary variable selection. One such thresholding rule is described Chapter 18 of Hastie *et al.* (2009): the t-statistics of all the variables should be graphically represented in a histogram, and if the histogram illustrates that the t_j-values are approximately normally distributed, then variables with $|t_j| > 2$ are considered as relevant. Using $|t_j| > 2$ corresponds to a significance level of approximately 5%. If a thresholding rule based on p-values is used, a variable is retained if its p-value is less than a pre-specified threshold. The candidate p-values will be dependent on the practical study used. In studies where there is uncertainty regarding the number of significant variables, a higher p-value threshold should be used to allow for richer models to be selected since it is not desirable to remove a larger number of variables in such an early stage of the analysis. Conversely, in studies where the researcher has a fairly good idea that only a smallish subset should be retained, then a smaller p-value threshold should be used. A very small p-value threshold implies that there is a strict selection criterion and thus only highly significant variables are retained.

Later in this thesis pairwise t-tests will be used for binary classification datasets to determine which of the original variables to retain, based on the p-value thresholding.

3.3.3 Score Statistic

Consider the response $Y \in \{0,1\}$ in a binary classification problem, and a single predictor variable X. Let $p(x) = P(Y = 1|X = x) = 1 - P(Y = 0|X = x)$. The log-likelihood of $Y|X = x$ is given by

$$Y \log[p(x)] + (1 - Y)\log[1 - p(x)] = Y \log\left[\frac{p(x)}{1 - p(x)}\right] + \log[1 - p(x)].$$

Put $\frac{p(x)}{1-p(x)} = e^{\beta x}$, then the log-likelihood becomes

$$l(\beta) = Y\beta x - \log\left[1 + e^{\beta x}\right].$$

The first and second derivatives of the log-likelihood are given by

$$\frac{\partial l}{\partial \beta} = Yx - x\left(1 + e^{\beta x}\right)^{-1},$$

$$\frac{\partial^2 l}{\partial \beta^2} = x^2\left(1 + e^{\beta x}\right)^{-2}.$$

The score statistic is given by

$$s(Y, x) = \frac{U^2(0)}{I(0)},$$

where $U(\beta) = \frac{\partial l}{\partial \beta}$, $I(\beta) = \frac{\partial^2 l}{\partial \beta^2}$. It follows that

$$s(Y, x) = -\frac{4\left[\left(Y - \frac{1}{2}\right)x\right]^2}{x^2}.$$

Now consider a sample $x_1, \dots, x_N$ of values of X, with corresponding values $Y_1, \dots, Y_N$ for Y. The full log-likelihood is given by

$$l(\beta) = \sum_{i=1}^{N} Y_i \log\left[\frac{p(x_i)}{1 - p(x_i)}\right] + \sum_{i=1}^{N} \log[1 - p(x_i)]$$

$$= \beta \sum_{i=1}^{N} Y_i x_i - \sum_{i=1}^{N} \log\left[1 + e^{\beta x_i}\right].$$

Taking first and second derivatives of this expression, and putting $\beta = 0$, the score statistic is obtained, *viz.*

$$s(\boldsymbol{Y}, \boldsymbol{x}) = -\frac{4\left[\sum_{i=1}^{N} x_i \left(Y_i - \frac{1}{2}\right)\right]^2}{\sum_{i=1}^{N} x_i^2}. \qquad (3.4)$$

This statistic can be used for variable screening, similar to the correlation coefficient. Consider the predictor variables $X_1, \dots, X_p$ and suppose N observations are available on each of these variables, with corresponding values $Y_1, \dots, Y_N$ of the binary response. For $j = 1, \dots, p$ compute

$$s_j \equiv s\left(\boldsymbol{Y}, \boldsymbol{x}_j\right) = -\frac{4\left[\sum_{i=1}^{N} x_{ij} \left(Y_i - \frac{1}{2}\right)\right]^2}{\sum_{i=1}^{N} x_{ij}^2}.$$

Let $s_{(1)} < s_{(2)} < \cdots < s_{(p)}$ be the ranked score statistic values. Given some threshold Δ, select/include all the predictor for which $s_j < \Delta$. Alternatively, given the number, r, of predictors which one would like to include, these are taken to be those corresponding to $s_{(1)}, \dots, s_{(r)}$.

It is interesting to note that there seems to be a relationship between the score statistic and the two-sample t-test statistic. Consider a single predictor X and the response Y. Let $\bar{x}_0$ be the mean of the values of X for which $Y = 0$, with a similar interpretation for $\bar{x}_1$. Also, assume that $N_0 = \sum_{i=1}^{N} \mathrm{Ind}(Y_i = 0) = {}^{N}\!/_{2}$, so that also $N_1 = {}^{N}\!/_{2}$. Then,

$$\bar{x}_1 - \bar{x}_0 = \frac{1}{N_1}\sum_{i=1}^{N} x_i\, Y_i - \frac{1}{N_0}\sum_{i=1}^{N} x_i\,(1 - Y_i)$$

$$= \frac{4}{N}\sum_{i=1}^{N} x_i\left(Y_i - \frac{1}{2}\right),$$

after some simplification and using the assumption $N_0 = N_1 = {}^{N}\!/_{2}$. It follows that

$$(\bar{x}_1 - \bar{x}_0)^2 = \frac{16}{N^2}\left[\sum_{i=1}^{N} x_i\left(Y_i - \frac{1}{2}\right)\right]^2.$$

This is very similar to the numerator in (3.4). Since the denominator seems to be only a scaling factor, variable screening using the score statistic may be equivalent to screening based on the two-sample t-test.

3.4 PRE-CONDITIONING FOR FEATURE SELECTION

Paul *et al.* (2008) proposed the method of “preconditioning” for feature selection and regression in high-dimensional problems. This is a method for variable selection that first estimates the regression function yielding values of a “preconditioned” response variable.

The method aims to solve two problems commonly faced in a high-dimensional data set separately, namely: finding a good predictor, $\hat{y}$, and determining a subset of predictors that play a role in predictions (Paul *et al.*, 2008:1596). Paul *et al.* (2008) propose pre-conditioning for feature selection as a two-step procedure; firstly, performing initial regression to obtain a preconditioned response using a primary approach such as supervised principal components and then the second step entails applying a standard model fitting procedure, such as forward stepwise selection or the LASSO, to the preconditioned response.

The first step in the procedure aims to denoise the outcome variable by computing the supervised component score for each observation in the training set (the threshold value is selected using CV). In Step 2, the LASSO is applied to the data set using $\hat{\boldsymbol{y}}_i$ as the outcome variable in place of $\boldsymbol{y}_i$. The authors explain that all features are used in the LASSO fit, not only those retained in the thresholding step in supervised principal components.

The idea behind denoising y_i and using the preconditioned response, $\hat{y}$, instead of y in the model selection step (in Step 2) is to alleviate the adverse effect that a large number of noise features has on the selection process (Paul *et al.,* 2008:1596). Furthermore, denoising of the outcome variable removes the issues of overfitting the LASSO to the outcome variable. Performing the preconditioning procedure with supervised principal components assumes that any important predictor variables, in terms of their regression coefficients, will also have a substantial marginal correlation with the outcome variable (Paul *et al.,* 2008:1600).

Preconditioning can be applied to classification problems and aims to obtain a good classifier as well as select a small set of good predictor features (such as a small set of uncorrelated features) for the classification procedure. Paul *et al.* (2008) explains that preconditioning in classification problems has the advantage over many classifiers such as SVMs that are only effective in finding a good separator for the classes but are not effective in refining the features into a smaller set of strong features.

The first step is to obtain the preconditioned probability estimates for each class, $\hat{p}_i$ for $i = 1,2, \dots, N$, of the predictor values using a trained classifier such as NSC.

Then the second step is to apply an appropriate selection procedure, such as the LASSO or forward stepwise regression, to an appropriate function of the preconditioned estimates from step one. In a binary classification problem, the logical choice is to use the logodds or logit quantity

$$\log\left[\frac{\hat{p}_i}{(1-\hat{p}_i)}\right].$$

Pre-conditioning can be applied in a variety of regression, classification and survival analysis settings, using different initial estimates (other than NSC and supervised principal components) and post-processors (as an alternative to forward stagewise regression and the LASSO). More detail on this method may be found in Paul *et al.* (2008).

3.5 SUMMARY

This chapter provided an introduction to variable selection, and the fundamental role it plays in high-dimensional data sets. This was followed by a description of traditional variable selection techniques, with application to binary classification problems. In high-dimensional problems, traditional variable selection methods, such as for example an all possible subsets approach, are no longer sufficient, mainly due to the large value of p. Therefore, modern procedures that are appropriate in high-dimensional settings were discussed briefly in this chapter. The modern variable selection techniques considered are: *least absolute shrinkage and selection operator*, *least angle regression,* f*orward stagewise regression, sure independence screening,* focusing on three different variable importance criteria for binary classification problems (*viz*., the correlation coefficient, the p-value of the two-sample t-test and the score statistic); and finally, *pre-conditioning for feature selection*.

The next chapter provides a description of various dimension reduction techniques which transform the original variables and which can be used as an alternative to or in combination with several variable selection approaches.

CHAPTER 4
DIMENSION REDUCTION

4.1 INTRODUCTION

The objective of statistical dimension reduction is to reduce the number of predictor variables, p, in a data set by replacing them with a much smaller number of transformed versions of the original predictor variables (Ye and Li, 2005). These transformations are typically linear combinations of the original variables. There are several advantages of dimension reduction, including reducing the danger of over-fitting (which is especially relevant in cases which have a large number of input variables), and offering the possibility of visualising the data in informative low-dimensional spaces. Also, classification of observations into two groups is not necessarily more accurate with a larger p value, since predictor variables that do not discriminate between the two groups may distort the result of the classification.

A popular class of dimension reduction techniques are projection methods, such as principal component analysis (PCA) and partial least squares (PLS), which attempt to determine directions in input space that result in small classification errors. This chapter will discuss PCA, supervised PCA and PLS as dimension reduction techniques to overcome the high-dimensionality in microarray data sets. Empirical investigations will focus on a comparison of ordinary principal components with supervised principal components.

4.2 PRINCIPAL COMPONENT ANALYSIS

Principal component analysis is a popular unsupervised data-processing and dimension reduction technique. PCA attempts to reduce the dimension of the data set by identifying and using informative linear combinations of the original predictors, instead of using the p predictors themselves. In order to reduce the dimension of the original data set, the number of linear combinations must be substantially less than p. According to Johnson and Wichern (2007:430) the aim of PCA is to find the linear combinations which maximise the variation in the data set.

Principal components are entirely dependent on the covariance (or correlation) matrix of the original variables. There are several ways to explain principal component analysis;

here the singular value decomposition (SVD) of the centred input matrix X is used to express the principal components of the variables in X, as explained by Gosh (2002). The SVD of the $N \times p$ matrix X (centred to have mean zero column vectors) has the form

$$X = UDV^T. \tag{4.1}$$

Here U is an $N \times p$ orthogonal matrix ($U^T U = I_p$) with the columns of U spanning the column space of X and V is a $p \times p$ orthogonal matrix ($V^T V = I_p$) whose columns span the row space of X. D is a $p \times \mathrm{p}$ diagonal matrix, with diagonal entries $d_1 \geq d_2 \geq \cdots \geq d_p \geq 0$ known as the singular values of X. Note that if one or more values of $d_j = 0$, then X is singular.

The sample covariance matrix is given by $S = \frac{1}{N} X^T X$, and from the SVD of X in (4.1) the eigen-decomposition of $X^T X$ can be written as

$$X^T X = VD^2 V^T,$$

which implies that

$$(X^T X)V = VD^2.$$

In the above equation, the columns $\boldsymbol{v}_1, \dots, \boldsymbol{v}_p$ of V are the eigenvectors of $X^T X$, with corresponding eigenvalues $d_1^2 \geq d_2^2 \geq \cdots \geq d_p^2 > 0$. The ordered sequence of eigenvectors, $\boldsymbol{v}_1, \dots, \boldsymbol{v}_p$, define the principal component directions of X and $d_1^2 \geq d_2^2 \geq \cdots \geq d_p^2$ represent the variances of the principal components (Hastie *et al.*, 2009:66). For any given data set there are at most $\min(N - 1, p)$ principal components.

The principal components (or principal component scores) of X are given by:

$$\boldsymbol{z}_j = X\boldsymbol{v}_j = UDV^T \boldsymbol{v}_j = d_j \boldsymbol{u}_j \quad \text{for } j = 1, \dots, p. \tag{4.2}$$

Here $\boldsymbol{z}_j$ is a vector of length N and the columns of U are the vectors $\boldsymbol{u}_j$ which are ordered to ensure that $d_1^2 \geq d_2^2 \geq \cdots \geq d_p^2$.

The first principal component of X is the normalised linear combination of $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_p$ that has the largest sample variance,

$$\boldsymbol{z}_1 = X\boldsymbol{v}_1 = v_{11}\boldsymbol{x}_1 + v_{12}\boldsymbol{x}_2 + \cdots + v_{1p}\boldsymbol{x}_p.$$

Here $\boldsymbol{v}_1$ is the first principal component direction which is normalised so that the sum of squares equals 1 ($\boldsymbol{v}_1{}^T\boldsymbol{v}_1$ =1 or $\sum_{j=1}^{p} v_{j1}^2 = 1$). The vector $\boldsymbol{v}_1$ defines the direction in the feature space along which the data varies the most.

Hastie *et al.* (2009:66) indicates that the sample variance of $\boldsymbol{z}_1$ is:

$$\mathrm{Var}(\boldsymbol{z}_1) = \mathrm{Var}(X\boldsymbol{v}_1) = \frac{1}{N}\boldsymbol{v}_1{}^T X^T X \boldsymbol{v}_1 = \frac{d_1^2}{N}.$$

It is seen that this variance depends directly on the largest eigenvalue. The second principal component, $\boldsymbol{z}_2$, is the linear combination of $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_p$ that has maximal variance out of all the linear combinations that are uncorrelated with $\boldsymbol{z}_1$. Using the constraint that $\boldsymbol{z}_2$ must be uncorrelated with $\boldsymbol{z}_1$ is equivalent to the constraint that the direction $\boldsymbol{v}_2$ must be orthogonal (perpendicular) to the direction $\boldsymbol{v}_1$ ($\boldsymbol{v}_1^T\boldsymbol{v}_2$ = 0). Similarly, the subsequent principal components $\boldsymbol{z}_j$ have maximum variance, $\frac{d_j^2}{N}$, subject to being orthogonal to the previous principal components. Hence, the small eigenvalues d_j^2 correspond to directions in the column space of X having small variance.

Summarising, the principal components, $\boldsymbol{z}_1, \dots, \boldsymbol{z}_M$ $(M < p)$, are uncorrelated and have variances proportional to the eigenvalues of Σ. The proportion of the total variation in the sample explained by the a^{th} principal component can be expressed as:

$$\frac{d_a^2}{d_1^2 + \cdots + d_p^2}. \tag{4.3}$$

Calculating the sum of (4.3) up to principal component m gives an expression for the proportion of the total variance that is explained by the first m principal components. This is denoted by ω_m, *i.e.*:

$$\omega_m = \frac{d_1^2 + d_2^2 + \cdots + d_m^2}{d_1^2 + d_2^2 + \cdots + d_p^2}. \tag{4.4}$$

PCA eliminates the problem of co-linearity between the variables in a data set, since the principal components are by definition uncorrelated (Johnson and Wichern, 2007:430). It is interesting to note that if the original predictor variables are highly correlated then the proportion of the total variance explained by the first m principal components is close to 1 even for small values of m.

An important step in PCA is to decide how many principal components to retain. Although there is no set rule to aid in this decision, there are several guidelines as proposed by Rencher (2002:434):

1. Retain sufficient components to account for a specified proportion of the total variance (the threshold value), say by letting $\omega_{frac} = 0.9$.
2. Exclude principal components whose eigenvalues are less than the overall average of the eigenvalues, calculated as:

$$\overline{d^2} = \sum_{j=1}^{p} \frac{d_j^2}{p}.$$

3. Represent the principal components in a scree plot, by plotting d_j^2 against j. A scree plot indicates the appropriate number of components to retain by an elbow (bend) in the plot which separates the large eigenvalues from the small eigenvalues.
4. Use significance tests to test if the "larger" principal components are indeed significant.

By requiring $\omega_m > \omega_{frac}$ (where ω_{frac} denotes the desired proportion of the total variance to be explained), the m principal components are determined and used to replace the original p predictor variables.

The first proposed method for determining how many principal components should be retained to effectively summarise the original data is the method implemented in this thesis. The main drawback associated with this method is the difficulty of determining the threshold value ω_{frac}. Setting this value too high runs the risk of retaining principal components that are sample or variable specific (Rencher, 2002:434). However, ω_{frac} will be set to 0.9 for the data exploration in this thesis.

When comparing dimension reduction by means of PCA to variable selection, the advantage of variable selection is that variables that do not discriminate between groups are removed and need not be observed for future purposes, as they are no longer considered important. However, in dimension reduction, all the variables remain in the model as linear combinations of all these variables are used. Therefore, variable selection techniques yield easier interpretation of the important predictors compared to dimension reduction by means of PCA.

PCA can also be used as a tool for data visualisation for both the observations and the variables, as it can be used to obtain a low-dimensional representation of the data that captures most of the information from the data, which is typically very complex in high-dimensional settings (James *et al.,* 2013:375)

In the experimental work reported later in the thesis, PCA will be implemented using the *preProcess()* function in the *caret* package in R. PCA will be applied to the original high-dimensional data sets as a dimension reduction step, and then KNN and SVM classification procedures will be implemented using the ordinary principal components, $\boldsymbol{z}_1, \dots, \boldsymbol{z}_M$ $(M < p)$, extracted from the data as inputs.

This section explained PCA using the SVD of $X = [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \cdots \quad \boldsymbol{x}_p]$; however, a modified weighted PCA could be performed implementing the SVD of $XD_\theta = [d_1(\theta)\boldsymbol{x}_1 \quad d_2(\theta)\boldsymbol{x}_2 \quad \cdots \quad d_p(\theta)\boldsymbol{x}_p]$. Supervised principal components fall within this framework, using $d_j(\theta) = \text{Ind}\left(\left|\hat{\beta}_j\right| \geq \theta\right)$. Supervised principal components are explained in detail in the following section.

4.3 SUPERVISED PCA

In contrast to PCA, supervised principal component analysis (SPCA) uses the response and the predictor variables to construct linear combinations of the input variables. SPCA performs PCA on a data-dependently identified subset of the original set of predictors, namely those which are strongly related to the response. There are different possibilities for measuring the strength of the relationship between the predictors and the response. The most commonly used measure in linear regression is the correlation coefficient and the standardised univariate regression coefficient (these two measures are equivalent), whilst in classification the correlation coefficient, two-sample t-test and the score statistic are commonly used measures. In SPCA the first step is to identify the predictors strongly related to the response by using one of these measures, followed by an ordinary PCA but using only these selected variables. Therefore, SPCA identifies linear combinations of the predictor variables that have both high variation and a significant relationship with the response variable. Supervised principal components are used in linear regression, classification, survival analysis, and other areas.

Bair and Tibshirani (2004) were the first to publish a detailed paper on the concept of supervised principal components. Other authors have presented related ideas, for example Ghosh (2002) pre-screened genes before extracting principal components to reduce computations. Note that in a later paper by Bair *et al.* (2006) the authors explain that SPCA performs selection based on scores to remove irrelevant sources of variations in an attempt to determine clusters of relevant predictors. Filtering out the noisy (irrelevant) predictors is the key to the good performance of SPCA.

Bair *et al.* (2006) describe the algorithm for SPCA in a regression setting as a four-step procedure. Firstly, the standardised univariate regression coefficients are computed for the response as a function of each feature separately. The standardised univariate regression coefficients are denoted by $\hat{\beta}_1, .., \hat{\beta}_p$ and are computed as follows:

$$\hat{\beta}_{\mathrm{j}} = \frac{\langle \boldsymbol{y}, \boldsymbol{x}_j \rangle}{\left\|\boldsymbol{x}_j\right\|^2}, \qquad j = 1, \dots, p.$$

Here $\left\|\boldsymbol{x}_j\right\|^{\mathbf{2}} = \boldsymbol{x}_j^T \boldsymbol{x}_j$**.** For example, regress Y on X_p to obtain $\hat{y} = \hat{\beta}_p X_p$ and consequently $|\hat{\beta}_p|$.

The second step is to form a reduced data matrix consisting of only those features whose univariate coefficients exceed a given threshold, θ, in absolute value. Therefore, SPCA focuses on a reduced matrix of input values, namely the matrix $XD_\theta = [d_1(\theta)\boldsymbol{x}_1 \quad d_2(\theta)\boldsymbol{x}_2 \quad \cdots \quad d_p(\theta)\boldsymbol{x}_p]$. Here D_θ denotes a diagonal matrix with diagonal entries $d_j(\theta)$ given by

$$d_j(\theta) = \mathrm{Ind}\left(|\hat{\beta}_j| \geq \theta\right).$$

From the above equation it is clear that XD_θ is a reduced data matrix, since if $d_j(\theta) = 0$, then the corresponding column $\boldsymbol{x}_j$ drops out. It is important to note that a specific value of θ leads to a set of indices

$$S_\theta = \{j : |\hat{\beta}_j| \geq \theta\},$$

where S_θ denotes the subset of $\{1,2, \dots, p\}$ that are selected using θ as the threshold value. Choosing the threshold value, θ, is discussed below.

The third step in SPCA is to compute the first (or first $m > 1$) principal components of the reduced data matrix. Let $X_\theta = XD_\theta$, then the SVD of X_θ is:

$$\boldsymbol{X}_\theta = \boldsymbol{U}_\theta \tilde{\boldsymbol{D}} \boldsymbol{V}_\theta^T .$$

Here $U_\theta = (\boldsymbol{u}_{\theta,1}, \boldsymbol{u}_{\theta,2}, \dots, \boldsymbol{u}_{\theta,m})$, where $\boldsymbol{u}_{\theta,1}$ is the first supervised principal component of X**,** while $\tilde{\boldsymbol{D}}$ is a diagonal matrix containing the singular values. Therefore, m supervised principal components are calculated in step three.

In the fourth and final step of SPCA, the principal component(s) computed in Step 3 are used for example in a regression model to predict the outcome. The regression model fit on the first supervised principal component, $\boldsymbol{u}_{\theta,1}$, with response $\boldsymbol{y}$ is

$$\hat{\boldsymbol{y}}^{spc,\vartheta} = \bar{\boldsymbol{y}} + \langle \boldsymbol{u}_{\theta,1}, \boldsymbol{y} \rangle \boldsymbol{u}_{\theta,1},$$

where $\bar{\boldsymbol{y}}$ denotes the mean of $\boldsymbol{y}$ and is the intercept in the regression model. The SPCA algorithm makes use of a fixed value of θ; however, in practical applications $\hat{\theta}$ will be used in the algorithm to represent θ where $\hat{\theta}$ is determined using for example CV.

Bair *et al.* (2006) explain that the values of θ and m should be chosen using cross-validation. Also, instead of using a single threshold value θ in Step 2 to perform hard-thresholding, soft-thresholding could be implemented using two different threshold values, θ_1 and θ_2. In this soft-thresholding approach, $\theta_1 < \theta_2$, and if

$$|\hat{\beta}_j| < \theta_1, \text{then } d_j(\boldsymbol{\theta}) = 0,$$

and if

$$\theta_1 \le |\hat{\beta}_j| < \theta_2, \text{ then } d_j(\boldsymbol{\theta}) = b_j,$$

where b_j is a weight that depends on $\max_j |\hat{\beta}_j|$. Finally, if

$$|\hat{\beta}_j| \ge \theta_2, \text{then } d_j(\boldsymbol{\theta}) = 1.$$

The above algorithm is for normal regression settings; however, certain adjustments can be made to apply supervised principal components to generalised regression settings such as the analysis of survival data or classification problems (see Bair *et al.*, 2006, for more details).

Instead of using the standardised univariate regression coefficients in Step 1, the SPCA procedure implemented in this thesis will compute two other measures which can be used in classification scenarios to rank the p predictor variables according to their ability and importance in distinguishing between two population groups. These measures are the correlation coefficient between a predictor and the response variable, and the two-sample t-test statistic or associated p-value. Therefore, two different versions of SPCA will be considered. The optimal threshold values for θ in Step 3 for both the correlation coefficient and the p-value in the two-sample t-test will be determined using cross-validation.

As explained in PCA, in this thesis the number of supervised principal components retained for each practical data set is determined by ω_{frac}, which is specified as 0.9 (the first supervised principal components that explain 90% of the variance in the data set will therefore be retained).

For the purpose of this thesis, the fourth and final step of the SPCA implements the KNN and SVM classification procedures on the supervised principal components data to classify each test case, thereby evaluating the performance of supervised PCA.

4.3.1 Comparison of PCA and SPCA

In this section an exploration into the difference between PCA and SPCA is performed in terms of their performance on binary high-dimensional practical data sets (*viz.* the colon and leukemia data sets). The colon and leukemia data sets are both microarray high-dimensional binary classification data sets which will be explained in detail in Chapter 5. In both PCA and SPCA the proportion of the total variance explained is set to 0.9. In Step 1 of the SPCA procedure the correlation coefficient between each predictor and the response variable is computed and correlation thresholding is used for feature extraction.

The following two figures show plots of the first two principal components (PC) against the first two supervised principal components (SPC) on the colon cancer data set and the leukemia data set respectively. In Figures 4.1 and 4.2, three different correlation coefficients are used in the SPCA thresholding. The correlation coefficient threshold values used are 0.1000, $\text{median}(0.1\,, 0.8 \times cor_{max})$ and 0.8× cor_{max}. Here cor_{max} is the maximum absolute correlation coefficient between any predictor and the response variable.

The two different population groups in the colon and leukemia data sets are denoted by red and green points respectively in Figures 4.1 and 4.2.

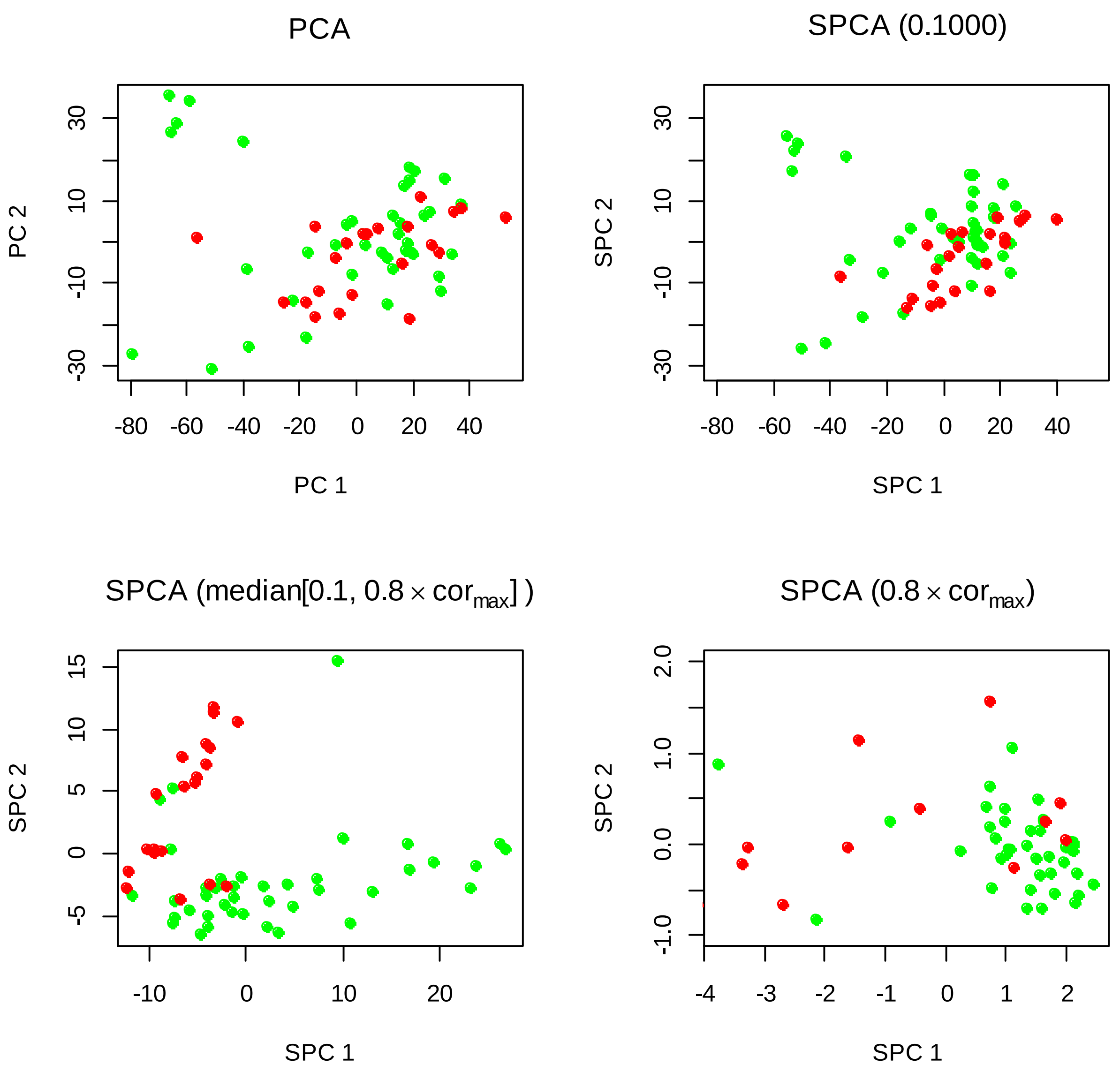

Figure 4.1: Comparison of the first two PCs and SPCs on the colon data set

The three correlation coefficient thresholds used in SPCA in Figure 4.1 are 0.1000 (top right), 0.3026 (bottom left) and 0.5053 (bottom right). Note that in Figure 4.1 the same scale is used for the PCs and the SPCs plotted in the panels in the top row; however, in the panels in the bottom row the scale for the axes vary depending on the SPC values.

From Figure 4.1 there appears to be a considerable overlap between the two population classes when these classes are represented in terms of the first two principal components (top left) and the first two SPCs using the correlation coefficient threshold equal to 0.1000 (top right). However, there appears to be less of an overlap between the population classes using the first two SPCs with a higher correlation coefficient threshold of 0.3026 (bottom left). Increasing the threshold further to a value of 0.5053 in SPCA results once again in an increase in overlap and difficulty in distinguishing between the two population groups. This is probably because at least some of the relevant input variables are eliminated when using a large correlation threshold. There also appears to be potential outliers in the two panels in the top row of Figure 4.1, as there is a cluster of observations in the top left corner of the graph belonging to the green population (P_2 is the tumorous colon tissue samples) that has much larger values for the second PC and SPC and smaller values for the first PC and SPC compared to the rest of the observations from this group.

The first two principal components plotted in the top left panel of Figure 4.1 are computed using all 2 000 genes in the colon data set. However, the first two SPCs plotted in the remaining panels of Figure 4.1 are each computed on a reduced subset of genes depending on the correlation coefficient threshold used when variables are eliminated. The SPC 1 and SPC 2 plotted in the top right panel of Figure 4.1 were computed using 1 201 genes, only those whose correlation coefficient with the response variable was larger than 0.1000 in absolute value. In the bottom left panel in Figure 4.1, SPC 1 and SPC 2 were computed using 218 genes, only those whose correlation coefficient with the response exceeded 0.3026 in absolute value. Finally, in the bottom right panel in Figure 4.1, SPC 1 and SPC 2 were computed using the strictest correlation thresholding value of 0.5053, resulting in only 8 genes remaining in the model (which is a 99.60% reduction in the 2 000 genes in the colon data set).

Although the information is not displayed in Figure 4.1, it is interesting to note that 90% of the variance in the data is explained by: 20 PCs, 20 SPCs using the correlation coefficient threshold equal to 0.1000, 16 SPCs with a correlation coefficient threshold equal to 0.3026 and only 3 SPCs using the correlation coefficient threshold equal to 0.5053.

From the comparison of PCA and SPCA on the colon data set displayed in Figure 4.1, it is evident that the first two SPC's using a correlation coefficient threshold of 0.3026 yields the least overlap between the two groups and is therefore likely to yield the highest classification accuracy.

The figure below compares PCA with SPCA on the leukemia data set, once again using three different correlation coefficient threshold values in SPCA (*viz.* 0.1000 (top right), 0.3939 (bottom left) and 0.6878 (bottom right)). Before discussing Figure 4.2, one should note that the first three panels in Figure 4.2 use the same scale, whilst the bottom right panel of Figure 4.2 is magnified to clearly illustrate the small SPCs values.

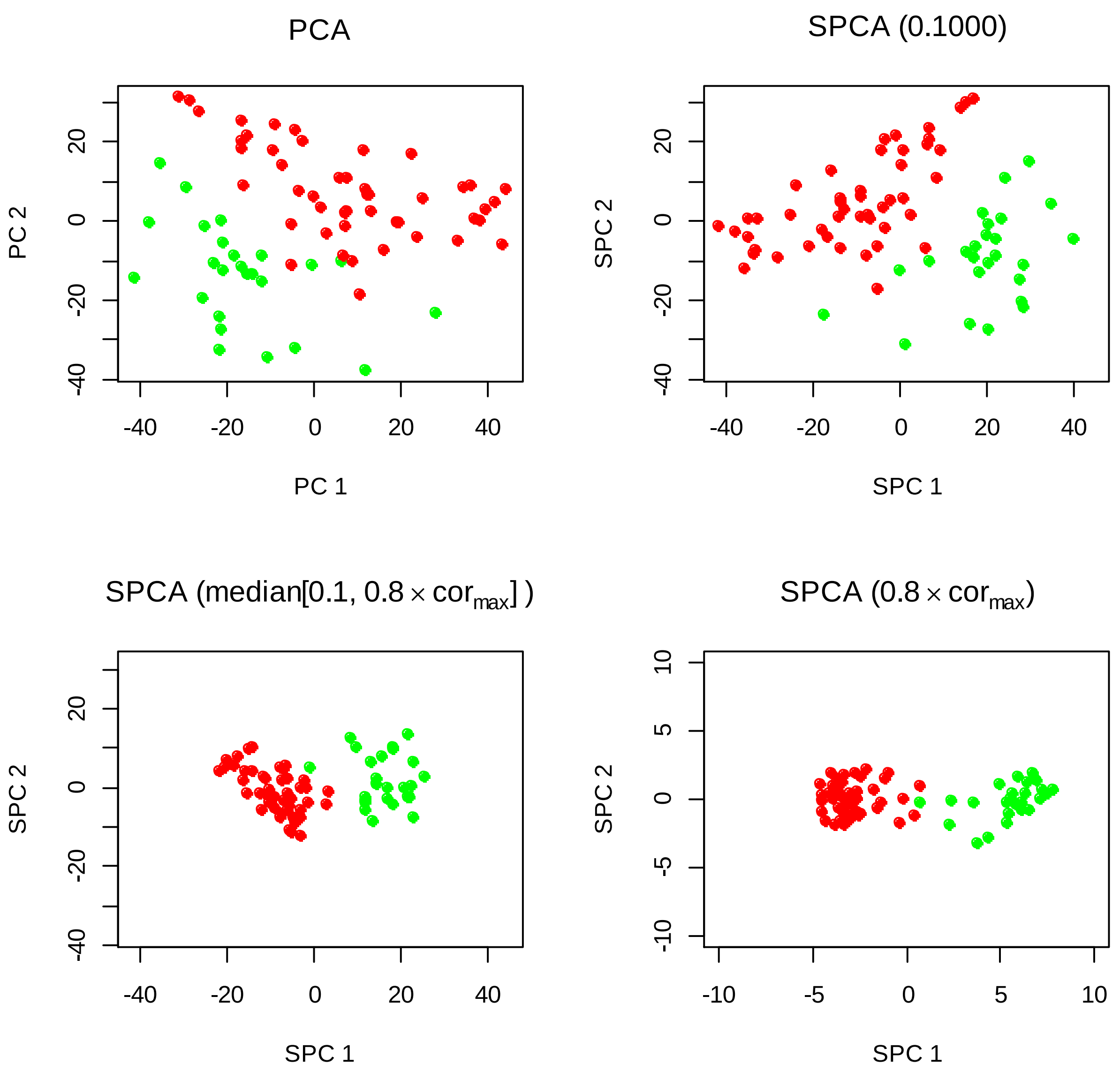

Figure 4.2: Comparison of the first two PCs and SPCs on the leukemia data set

There is less overlap between the two population groups in the first two PCs for the leukemia data set plotted in the top left panel of Figure 4.2 than in Figure 4.1 for the colon data set. There appears to be no outliers in any of the four panels in Figure 4.2. Figure 4.2 illustrates that as the correlation coefficient threshold used in SPCA increases, there is less overlap between the AML and ALL groups in the leukemia data set. Note that PCA could be seen as implementing SPCA with a correlation coefficient threshold equal to 0 in absolute value, thus including all 3 571 genes in the computation of the SPCs.

As in Figure 4.1, the first two principal components plotted in the top left panel of Figure 4.2 are computed using all 3 571 genes in the leukemia data set. However, the first two supervised principal components plotted in the top right panel of Figure 4.2 were computed using the subset of 2 546 genes whose correlation coefficient with the response variable is larger than 0.1000 in absolute value. In the bottom left panel the first two SPCs computed on the subset of 555 genes whose correlation with Y is greater than 0.3939 in absolute value, are illustrated. Finally, performing SPCA with a correlation coefficient threshold of 0.6878 on the leukemia data set resulted in the first two SPCs being computed on only 28 genes.

Although the information is not displayed in Figure 4.2, it is once again interesting to note that 90% of the total variance in the data is explained by: 52 PCs, 50 SPCs using the correlation coefficient threshold equal to 0.1000, 41 SPCs with a correlation coefficient threshold equal to 0.3939 and only 10 SPCs using the correlation coefficient threshold equal to 0.6878.

From the comparison of PCA and SPCA on the leukemia data set displayed in Figure 4.2, it is evident that the first two SPCs using the largest correlation coefficient threshold of 0.6878 yields the least overlap between the two groups and is likely to yield the most accurate classification for the leukemia data set.

Therefore, the comparison of PCA and SPCA displayed in Figure 4.2 indicates that in general the larger the correlation coefficient threshold used in SPCA the higher the dispersion between two populations groups when plotting the first two SPCs. However, the optimal correlation thresholding value to use in SPCA is dependent on the data set. As shown in Figure 4.1, the median correlation coefficient value of 0.3026 results in the smallest overlap between the two groups in the colon data set, whilst in Figure 4.2 the maximum correlation coefficient threshold considered (0.6878) yields the least overlap between the AML and ALL population groups in the leukemia data set.

4.4 PARTIAL LEAST SQUARES

Partial least squares is a well-known dimension reduction technique, introduced by Wold (1975). As in PCA, PLS constructs linear combinations of the original predictors, $X_1, \ldots, X_p$, but in a supervised manner, *i.e.* using the response, Y. The PLS procedure attempts to find uncorrelated linear combinations of $X_1, .., X_p$ producing a sequence of derived, orthogonal inputs or directions, $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_M$, commonly known as latent variables (Boulesteix, 2004). Furthermore, PLS aims to construct $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_M$ which simultaneously have high covariance with Y and high variance. The number of PLS directions, $M \leq p$, is a tuning parameter that is typically chosen by cross-validation. The number of latent variables is normally substantially smaller than N, thus overcoming the small sample size problem in for example genetic data sets and allowing simple classification techniques to be applied to the latent variables obtained in PLS. PLS is frequently used as a dimension reduction method in classification problems with both high-dimensional data or data which has the added complication of multicollinearity amongst the predictors $X_1, \ldots, X_p$. PLS was traditionally designed for a continuous response variable and has only relatively recently been adapted to deal with classification for high-dimensional data sets (Chung and Keleş, 2010).

PLS regression is viewed as a supervised alternative to principal component regression (James *et al.*, 2013:237). Hastie *et al.* (2009:82) explain that while PCA retains M high-variance directions and discards the rest, it can be shown that PLS tends to shrink the low-variance directions but can actually inflate some the of higher variance directions. Therefore, PLS may be unstable in terms of prediction error.

PLS has several advantageous properties, namely: it can be applied to both a univariate and multivariate response variable, is computationally efficient and it allows for graphical representation of a high-dimensional data set through its projection into a lower dimensional space (Chung and Keleş, 2010).

Hastie *et al.* (2009:680) state that PLS shrinks noisy irrelevant predictors but does not remove them and as a result a large number of noisy predictors can contaminate the predictions. Therefore, although PLS can be applied to a high-dimensional data set, classification procedures frequently make use of variable selection as a pre-processing step before fitting PLS to the data (Chung and Keleş, 2010). The two-sample t-test statistic explained previously in Section 3.3.2 is a commonly used pre-selection approach for binary classification.

For more information on PLS please refer to the paper by Chung and Keleş (2010) as well as Boulesteix (2004) and the references therein.

4.5 SUMMARY

This chapter explores a class of dimension reduction techniques which transform the original predictor variables into derived inputs. The three dimension reduction techniques considered in this chapter are: *principal component analysis*, a popular unsupervised learning technique which derives a low-dimensional set of features from a potentially large set of variables; *supervised principal component analysis*, a supervised learning technique which is a modification of traditional PCA and uses both the response and the predictor variables to construct linear combinations of the input variables; and finally, *partial least squares*, a technique which constructs linear combinations of the original predictors in a supervised manner. The first two of these dimension reduction techniques will be used in the empirical study reported in Chapters 6. Details will be provided of an empirical investigation that was performed on two data sets for binary classification to compare ordinary principal components with supervised principal components.

In the next chapter the data sets which are used in the empirical study are introduced. Details are also provided of the empirical work that was undertaken to determine the optimal thresholding values for each base classifier.

CHAPTER 5
EMPIRICAL WORK

5.1 INTRODUCTION

This chapter begins with a brief description and discussion of microarray data, in particular the technology used in microarray data set, its role in medical research and the problems it faces. This is followed by a description of the practical data sets considered, as well as the synthetic microarray data sets simulated in the thesis. A brief overview of the classification procedures as well as details regarding their implementation are described in Section 5.5.

The focus of the final section of this chapter is a discussion regarding determining the tuning (threshold) parameters values for the correlation thresholding variable selection procedure, the two-sample t-test variable selection procedure, and finally, the NSC variable selection procedures. The concept of leave-one-out CV is introduced and used to explore the role of the tuning parameters in the classification procedures in detail. In this section, substantial numerical evidence was presented, illustrating the behaviour of different candidate values used in the three different VS procedures for the KNN, SVM and NSC classifiers on microarray data sets.

5.2 MICROARRAY DATA

The focus in this part of the thesis will be on classification in the context of DNA microarray gene expression analysis, with the goal of classifying and predicting the diagnostic category of a sample based on its gene expression profile. DNA stands for deoxyribonucleic acid, and is defined as the self-replicating material which is present in nearly all living organisms and which is the basic material that makes up human chromosomes (Oxford Dictionaries, 2017, s.v. ‘DNA’). DNA microarrays are glass microscope slides that are printed with thousands of tiny spots in defined positions, with each spot containing a known DNA sequence of genes. DNA microarrays are used to measure the gene expression of a cell by measuring the amount of mRNA (messenger ribonucleic acid) present in that gene. Buhler (2002) describes mRNA as the type of RNA which codes for proteins. DNA microarrays were invented in the 1990s and are considered a breakthrough technology in biology, facilitating the quantitative study of thousands of genes simultaneously from a single sample of cells. Subsequently, microarrays have been the most popular technology for large-scale studies of gene expressions as they are readily affordable by many laboratories (Zhao *et al.,* 2014:1).

An $N \times p$ gene expression data matrix is the output of N microarray experiments or biological samples analysing p different genes. The symbols $X_1, X_2, .., X_p$ denote the gene expression measurements. The standard experimental protocol for obtaining gene expression data sets is described schematically in the figure below.

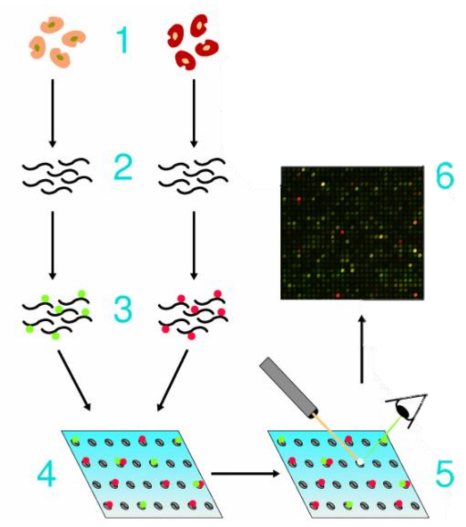

Figure 5.1: Schematic of the steps in an experimental protocol to study differential expression of genes

Source: Buhler, 2002.

The six steps in Figure 5.1 are explained in detail by Buhler (2002). The first step is choosing the cell sample populations, for example in Figure 5.1, one sample is taken from reference cells (orange) and another from target cells (red). Here, the reference sample could represent normal cells while the target sample represents tumorous cells. In Step 2, mRNA is isolated and extracted from the two cell samples. Buhler (2002) explains that mRNA is prone to being destroyed and degraded, therefore mRNA is converted into the more stable complementary DNA (cDNA) form using enzyme reverse transcriptase to prevent experimental samples from being lost. In Step 3, the cDNA is labelled with fluorescent markers to detect the cDNA of the two samples bound to the microarray. The reference sample cDNA is labelled with green dye (which replaces the orange used in Step 1) and the target sample cDNA with red dye. These are represented by the red and green circles attached to the cDNA in Step 3.

Following this step, the two cDNA samples are hybridized onto the same microarray slide, which holds hundreds or thousands of spots, each containing a different DNA sequence.

The term “hybridize” refers to the binding of complementary pairs of DNA molecules (Buhler, 2002). The labelled cDNA from both samples are mixed together and placed onto a DNA microarray slide, where each gene represented by a cDNA molecule will hybridize to the spot containing its cDNA sequence on the microarray. The amount of cDNA bound to a spot is directly proportional to the initial number of RNA molecules present for that gene in both samples.

Step 5 involves scanning the hybridized microarray using a laser at suitable wavelengths to detect both red and green dyes. The laser scanner measures the fluorescence of each spot on the hybridized array and the results of the one colour are then superimposed over those of the other.

The final step is to interpret the scanned image consisting of thousands of different coloured dots shown in Step 6 in Figure 5.1. The fluorescence intensity for each spot on the array is related to the amount of cDNA in the sample (and therefore also the mRNA level in the cell) for that gene (Howard Hughes Medical Institute, 2017). In the image in Step 6, if a gene has a high expression level in the target cell but not in the reference cell then it is represented by a dot that glows bright red. Conversely, a dot that glows bright green represents a high gene expression level in the reference cell but not in the target cell. The Howard Hughes Medical Institute (2017) explain that if a gene is expressed to the same extent in both samples it will be represented by a yellow spot indicating the combination of the red and green light. Finally, if the gene is not expressed in either sample then the spot is black.

Before the microarray data obtained in Step 6 in Figure 5.1 can be analysed, the different dot colours need to be converted into numbers that represent the intensity of the red and green dye. The intensities of RNA hybridized at each spot provided by the microarray image in Figure 5.1 can be quantified by computing the log of the expression ratio. For more information regarding the expression ratio, refer to the paper by Babu (2004).

The above process results in thousands of numbers measuring the expression level of each gene in the target sample relative to the reference sample. A positive value indicates a higher gene expression level in the target sample versus the reference sample, indicating that the gene was stimulated to make more mRNA by tumour formation (Howard Hughes Medical Institute, 2017). Conversely, negative values indicate higher gene expressions in the reference sample versus the target sample.

The most commonly used and important application of microarray technology is tumour diagnosis through classification (Boulesteix, 2004). Tibshirani *et al.* (2001:6567) state that there are two main reasons why classification of microarrays is challenging. Firstly, there are a large number of inputs (genes) from which to predict classes and a relatively small number of samples leading to high-dimensional data analysis problems (gene expression data are usually wide, implying $p \gg N$). Secondly, it is important to identify which genes make the highest contribution to the classifier, as not all the genes used in the expression profile are informative and many of them are redundant. To overcome the curse of dimensionality in microarray data, it is crucial to identify the genes contributing the most to classification as it can aid biological understanding of the disease process as well as playing a vital role in the development of clinical tests for early diagnosis (Tibshirani *et al.,* 2003:104). Precise identification of tumours is very important for treatment and diagnosis.

RNA sequencing (RNA-seq) is a recently developed approach to transcript profiling which makes use of deep-sequencing technologies (Wang *et al.,* 2009:57). Recently, RNA-seq has been used as an alternative to microarrays. Zhao *et al.* (2014:1) state that although RNA-seq has benefits compared to microarrays, microarrays are still the more common choice of researchers when conducting transcriptional profiling experiments.

5.3 PRACTICAL DATA SETS

Three different practical classification data sets will be examined in this thesis, namely the colon cancer, leukemia and SRBCT data sets. All of these are wide microarray data sets, with the number of genes, *i.e.* the number of predictor variables, exceeding the number of observations ($p \gg N$). The first two data sets contain binary classification data and the third represents a multi-class classification problem. Details on the data sets are given in the sections which follow.

5.3.1 Colon cancer data set

The first practical data set which is considered in this thesis is the colon cancer microarray data set (henceforth referred to as the colon data set) which was originally studied by Alon *et al.* (1999) and subsequently investigated by Guyon *et al.* (2002), Weston *et al.* (2003), and Rakotomamonjy (2003). The data set has gene expression samples that were analysed with an Affymetrix Oligonucleotide array. It should be noted that a special pre-processing of the data is performed by these authors analysing the colon cancer data set, which entails log-transforming the input variables, standardizing the resulting values and applying a so-called "squashing function", $f(x) = c\arctan\left(\frac{x}{c}\right)$, to diminish the effect of outliers. This thesis also applies this pre-processing to the colon cancer data set.

Colon tissues were biopsied from patients, and then tested to determine if adenocarcinoma was present in the colon tissue. The American Cancer Society (2014) defines adenocarcinoma as a type of cancerous tumour that forms in the cells that produce mucus to lubricate the inside of the colon and rectum, and is the most common type of colon cancer. The colon data set was formed using 62 samples of biopsied colon epithelial cells from potential colon cancer patients, which were reported as tumorous colon tissue if adenocarcinoma was present in the tissue samples; otherwise they were reported as normal colon tissue.

Table 5.1: Summary of the colon cancer microarray data set

Tumorous colon tissue	**Normal colon tissue**	**Total number of colon tissue**
40	22	62

Source: Alon *et al.*, 1999

The colon data set consists of $N = 62$ colon tissue samples and $p =$ 2 000 gene expression levels analysed in the colon tissue samples. The response variable, Y, in the colon data set represents the normal colon tissues and tumorous colon tissue, and originally consisted of 1's and -1's; however, for the purpose of this thesis the response variable was transformed to be made up of 0's and 1's. There are therefore two population groups, *i.e.* $G = 2$, with 22 observations in Group 1, which is the normal colon tissue represented by 0's in $\boldsymbol{y}$ and 40 observations in Group 2 (tumorous colon tissue represented by 1's in $\boldsymbol{y}$). Classification procedures will be applied to the colon data set using the gene expression levels to classify future colon tissue as being cancerous or non-cancerous. The colon cancer data set is considered a high error rate data set (Boulesteix, 2004:16).

5.3.2 Leukemia data set

The second data set considered is the leukemia data set which was introduced by Golub *et al.* (1999). This data set contains the gene expression levels for $N = 72$ leukemia mRNA samples which are split into two classes: acute lymphoblastic leukemia (ALL) and acute myeloid leukemia (AML). The samples of cells were obtained from bone marrow or peripheral blood of 72 individuals with leukemia, and the gene expression levels for each sample were measured using Affymetrix (one-colour) high-density oligonucleotide arrays containing probes for 6817 human genes (Dudoit *et al*., 2002:80). The arrays had 7 129 locations but only 6 817 contained probes relative to human genes and the rest are controls and replicates.

It should be noted that there is confusion across papers regarding the exact number of input variables as some authors quote that there are 6 817 gene-expression measurements, while others quote 7 129 gene-expression measurements. However, in this thesis the leukemia data set is used as obtained in the *varbvs* package in R as described below.

According to Golub *et al.* (1999:531) it is important to distinguish ALL from AML in order to successfully treat leukemia patients. This is the case since chemotherapy regimens for ALL generally contain corticosteroids, vincristine, methotrexate, and L-asparaginase, whereas most AML regimens rely on a backbone of daunorubicin and cytarabine. While ALL patients treated with AML therapy can achieve remissions (and vice versa), the probability of the patients being cured decreases and the patient is exposed to unwarranted toxicities resulting from application of the incorrect treatment regimen (Golub *et al.,*1999:531).

The data set consists of 47 ALL-leukemia samples which form Group 1 (denoted by $Y = 0$) and 25 cases in Group 2, which is the AML-leukemia samples (represented by $Y = 1$). Note that in the original leukemia data set, the ALL class is further divided into T-cell or B-cell, but this is not considered in this thesis.

The original leukemia data set undergoes a pre-processing procedure described in Dudoit *et al.* (2002) to find the genes with the most variability and exclude low variance genes, resulting in only $p = 3\ 571$ genes remaining in the reduced data set. The gene expression levels in the leukemia data set were normalised before undergoing pre-processing to ensure comparability across the samples.

Dudoit *et al*. (2002) describe three pre-processing steps which were applied to the normalized matrix of intensity values available on the website (after pooling the 38 mRNA samples from the learning set and the 34 mRNA samples from the test set):

(a) thresholding: floor of 100 and ceiling of 16 000;

(b) filtering: exclusion of genes with $\frac{max}{min} \leq 5$ or $(max - \min) \leq 500$, where max and min refer to the maximum and minimum intensities for a particular gene across the 72 mRNA samples;

(c) base 10 logarithmic transformation and standardisation.

Step (b) in the pre-process is implemented to eliminate genes presenting insufficient variation across samples in the analysis. This thesis follows the protocol in Dudoit *et al.* (2002) exactly and the reduced leukemia data set was obtained from the *varbvs* package in *R*.

Golub *et al.* (1999) state that it is possible to achieve excellent classification accuracy on this data set even with quite trivial methods and it is therefore considered a low error rate data set.

5.3.3 SRBCT data set

The SRBCT data set consists of $p = 2\,308$ gene expression levels for $N = 83$ small round blue cell tumours (SRBCT) found in children. This data set is presented in Khan *et al.* (2001). The expression levels for the genes were obtained from glass-slide complementary DNA (cDNA) microarrays (Tibshirani *et al.*, 2010:6567). Poplack and Pizzo (1997) explain that SBRCTs of childhood, which include neuroblastoma, rhabdomyosarcoma, non-Hodgkin's lymphoma, and the Ewing's family of tumours, are so called because of their similar appearance on routine histology. The types of SRBCTs all have a very similar appearance but belong to four distinct categories which makes correct clinical diagnosis using light microscopy extremely challenging. However, it is crucial to accurately diagnose the SRBCT as treatment options, responses to therapy and prognoses range widely depending on the diagnosis (Khan *et al*., 2001).

The SRBCT data set is a multiclass data set and SRBCT patients belong to one of the following four childhood tumour classes: BL (Burkitt lymphoma), EWS (Ewing Family of Tumors), NB (Neuroblastoma) and RMS (Rhabdomyosarcoma), as shown in Table 5.2.

Table 5.2: Summary of the SRBCT microarray data set

BL	**EWS**	**NB**	**RMS**
11	29	18	25

Source: Khan *et al.,* 2001

The response variable, $Y \in \{1,2,3,4\}$, corresponds to the four tumour classes, namely BL, EWS, NB and RMS respectively. The original data set consists of 63 training samples and 25 test samples and is available at https://statweb.stanford.edu/~tibs/ElemStatLearn/data.html. However, out of the 25 test samples, the response value of five samples were missing, as they were not SRBCT. These five cases were removed for the purpose of the thesis.

There is some confusion regarding the group names and their respective labels across the papers which investigate the SRBCT data set; however, in this thesis the names and labels for the SRBCT data set follow those used in general agreement.

The training and test set were combined to create one data set consisting of 83 SRBCT patients. The classification models built on the SRBCT data set attempt to distinguish between these four tumours based on gene expression values.

5.4 SIMULATED DATA SETS

The different classification procedures studied in this thesis were also applied to several synthetically generated data sets. More specifically, three high-dimensional binary microarray data sets were generated using the basic outline of the binary classification simulation example described by Tibshirani *et al.* (2003:111). All three of these simulated data sets consisted of standard normal expression data for $p = 1\,000$ genes and $N = 40$ samples, with 20 samples in each of the two population classes, *i.e.* $N_1 = 20$ sample cases from P_1 and $N_2 = 20$ sample cases from P_2. A distinction was made between two sets of genes: it was assumed that p_{rel} of the p genes were relevant in the sense of coming from different distributions for the two populations. The remaining $p - p_{rel}$ variables are referred to as irrelevant. The value of p_{rel} was kept at a constant value of 100 for each simulated data set. The three data sets differed in terms of the distributions from which values of these genes were generated. In all three cases the input variables corresponding to the relevant genes were generated from standard Gaussian distributions in the case of P_1. For P_2 these values were generated from a location shifted Gaussian distribution for the first data set investigated, from a scale shifted Gaussian distribution for the second data set, and from a location and scale shifted distribution for the third data set.

The following figure illustrates the basic layout of the gene expression levels for the three simulated data sets with p genes, p_{rel} relative genes and balanced samples from the two populations, *i.e.* $N_1 = N_2$, where $p > N = N_1 + N_2$. In Figure 5.2, the blue shaded regions represent the expressions levels that have an $N(0,1)$ distribution. The orange shaded region represents the gene expression levels whose distribution differs from $N(0,1)$.

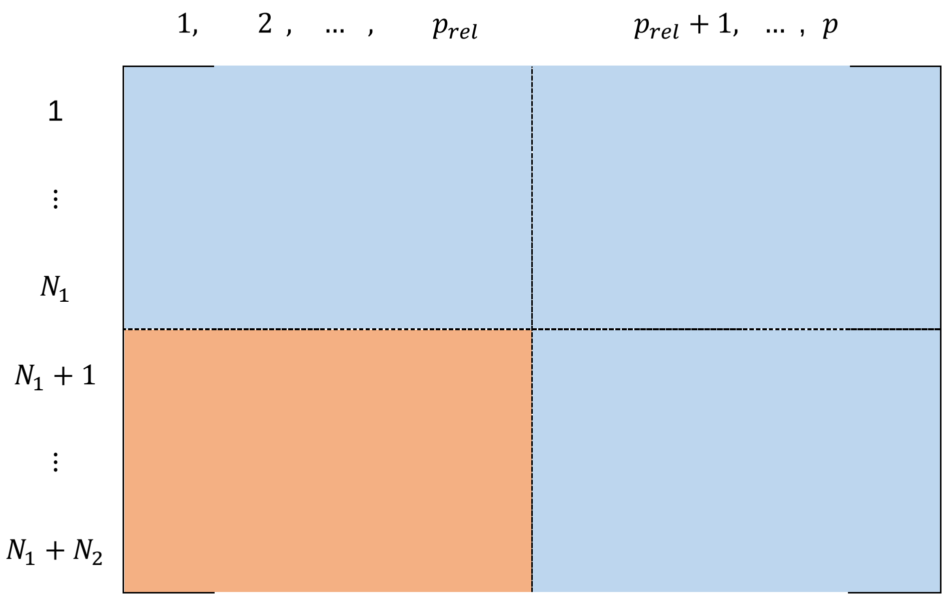

Figure 5.2: Schematic of the simulated data sets

From Figure 5.2 it is evident that for the sample cases from P_1, the gene expression levels for all of the p genes have the same distribution. However, in the case of P_2, the distribution of the expression levels for the p_{rel} relevant genes differ from that of the remaining $p - p_{rel}$ genes. Clearly, the three simulated data sets differ in terms of the distribution of gene expression levels only for the orange shaded region in Figure 5.2.

Note that all three simulated data sets are balanced $(N_1 = N_2)$, since the primary focus of this thesis is not to investigate the effect of balanced versus unbalanced data sets. However, additional research should be conducted to determine the influence of balanced and unbalanced data sets on the relative performances of the classification procedures.

The first simulated data set (referred to as Sim1) adds a constant to the generated values of the first p_{rel} genes in P_2. Let x_{ij} represent the expression level of the j^{th}gene for the i^{th} sample (patient). The values in Sim1 were generated as follows:

$$x_{ij} \sim \begin{cases} N(0,1) & i = 1, \dots, N_1 \quad and \;\; j = 1, \dots, p \\ N(0,1) & i = N_1 + 1, \dots, N \quad and \;\; j = p_{rel} + 1, \dots, p \\ N(0.5, 1) & i = N_1 + 1, \dots, N \quad and \;\; j = 1, \dots, p_{rel}\,. \end{cases} \tag{5.1}$$

Note that in (5.1), generating values from $N(0.5,1)$ is equivalent to adding 0.5 to values generated from $N(0,1)$. Hence, the first p_{rel} of the p =1 000 genes are differentially expressed in the two classes by a constant amount. Note that there is no scale difference between P_1 and P_2 in Sim1.

In the second simulated data set (referred to as Sim2), P_1 and P_2 differ with respect to scale (variance-covariance structure) *i.e.*, here the mean vectors are identical. Let x_{ij} once again represent the expression level of the j^{th} gene for the i^{th} sample (patient). For Sim2 these values were generated as follows:

$$x_{ij} \sim \begin{cases} N(0,1) & i = 1, \dots, N_1 \quad and \;\; j = 1, \dots, p \\ N(0,1) & i = N_1 + 1, \dots, N \quad and \;\; j = p_{rel} + 1, \dots, p \\ N(0, 0.5^2) & i = N_1 + 1, \dots, N \quad and \;\; j = 1, \dots, p_{rel}. \end{cases} \tag{5.2}$$

It is clear from (5.2) that now the first p_{rel} of the p = 1 000 genes have a smaller variance in P_2 than in P_1, while the two populations do not differ with respect to location.

Finally, in the third simulated data set (denoted by Sim3) the distribution of the first p_{rel} genes in P_2 differs with respect to both location and scale from this distribution in P_1. Let x_{ij} be as before, then the values in Sim3 were generated as follows:

$$x_{ij} \sim \begin{cases} N(0,1) & i = 1, \dots, N_1 \quad and \;\; j = 1, \dots, p \\ N(0,1) & i = N_1 + 1, \dots, N \quad and \;\; j = p_{rel} + 1, \dots, p \\ N(0.5, 0.5^2) & i = N_1 + 1, \dots, N \quad and \;\; j = 1, \dots, p_{rel} \end{cases} \tag{5.3}$$

The *R* programmes used to generate the three synthetic data sets are included in Appendix B.

5.5 CLASSIFICATION PROCEDURES

This section provides a brief description of the implementation details of the classification procedures considered in this thesis.

It is important to note that the values in the investigated data sets were only standardised if it is statistically necessary for a specific dimension reduction and classification procedure. Therefore, the binary data sets were standardised before applying the SVM classification procedure. Standardisation was carried out by computing means and standard deviations on the training data set and then using these quantities to standardise the values in both the training and test sets.

Hastie *et al.* (2009:79) state that principal components depend on the scaling of the inputs, implying that data sets should be standardised before performing PCA or SPCA as a dimension reduction technique. Note that standardisation is done automatically in the *preProcess*() function from the *caret* package in R which was used to perform PCA and SPCA.

The following diagram provides a summary of the classification procedures applied to the binary data sets:

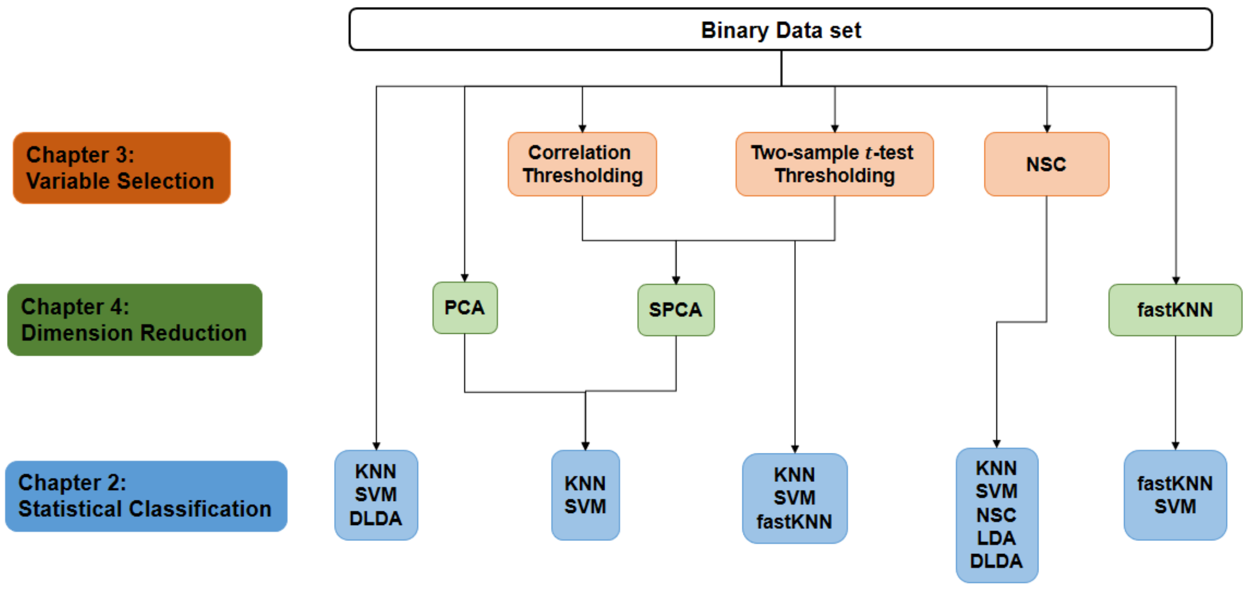
Binary Data set
Chapter 3: Variable Selection
Correlation Thresholding
Two-sample t-test Thresholding
NSC
Chapter 4: Dimension Reduction
PCA
SPCA
fastKNN
Chapter 2: Statistical Classification
KNN SVM DLDA
KNN SVM
KNN SVM fastKNN
KNN SVM NSC LDA DLDA
fastKNN SVM

Figure 5.3: Summary of the binary classification procedures

Figure 5.3 summarises the 21 different classification procedures that were applied to the binary data sets, both practical and simulated. The statistical classification procedures in Figure 5.3 can be divided according to the three main classifiers (and their extensions), namely: KNN (fastKNN), SVM and LDA (DLDA and NSC). Figure 5.3 also displays the three variable selection and dimension reduction techniques which are implemented in the thesis. Note that the binary data sets may undergo both VS and dimension reduction as shown in Figure 5.3.

From Figure 5.3 it is clear that LDA and its extensions are applied in four different versions, namely:

1. Ordinary LDA using the reduced set of inputs computed using NSC as a variable selection method
2. The NSC classification procedure (which entails an extension of DLDA)
3. DLDA on all p original variables
4. DLDA using the reduced set of inputs computed using NSC as a VS method.

The following diagram provides a summary of the classification procedures applied to the multi-class SRBCT data set:

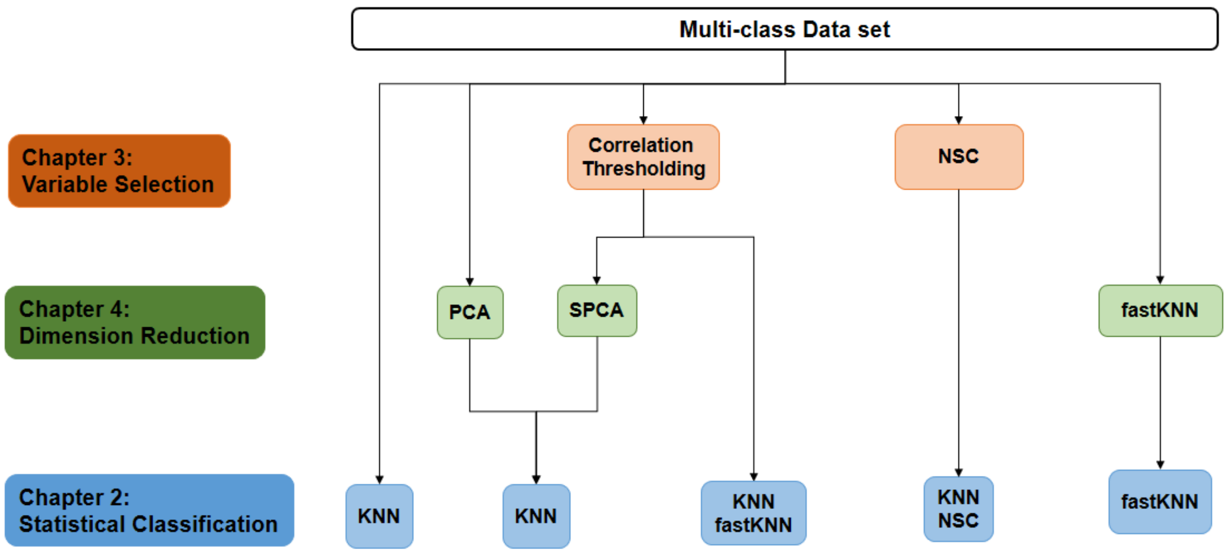
Multi-class Data set
Chapter 3: Variable Selection
Correlation Thresholding
NSC
Chapter 4: Dimension Reduction
PCA
SPCA
fastKNN
Chapter 2: Statistical Classification
KNN
KNN
KNN fastKNN
KNN NSC
fastKNN

Figure 5.4: Summary of the multi-class classification procedures

Figure 5.4 clearly illustrates that only three classification procedures are applied to the multi-class SRBCT data set, namely KNN, fastKNN and the NSC classifier. Figure 5.4 displays only two VS techniques, correlation thresholding and the NSC procedure. Although two-sample t-test thresholding could be implemented on the multi-class data set by considering all $\binom{4}{2}$ pairs of classes, it is not considered in this thesis. Furthermore, the optimal correlation threshold value is only determined for the KNN classifier while the optimal NSC Δ value is only determined for the NSC classifier and not for the KNN classifier.

The following section describes the manner in which the various feature selection, dimension reduction and classification procedures displayed in Figures 5.3 and 5.4 were implemented using R.

5.5.1 KNN and fastKNN

In this thesis, the *knn()* function from the *class* package in R was used to implement the KNN classification procedure. The KNN classification procedure was applied using three different numbers of nearest neighbours, namely $K \in \{1, 3, 5\}$. This was done to investigate the possible effect of this tuning parameter on the classification performance of KNN. For each case in the test set, the *knn()* function determines the K nearest cases (in terms of Euclidean distance) in the training set, and the classification of the test case is decided by majority voting, with ties broken at random. The *use.all* argument in the *knn()* function controls the handling of ties. In this thesis *use.all = FALSE* was used, which implies that if there were ties then a random selection of distances equal to the K^{th} is chosen to come to a decision.

Finally, for the binary scenario it is important to note that in the *knn* package in R, the procedure outputs Y as a value 1 or 2, instead of a 0 or a 1. Therefore the output has to be transformed to a 0 or a 1. However, in the multi-class scenario this transformation is not necessary.

As mentioned above, fastKNN was applied as both a classification and as a feature extraction procedure. The fastKNN classifier was applied using the *fastknn()* function in the *fastknn* package in R and the same three values of K as in the KNN classifier were used, *i.e.* $K \in \{1, 3, 5\}$. As with the *knn()* function, the *fastknn()* function outputs Y (*$class*) as a 1 or a 2 instead of a 0 or a 1. Therefore the output has to be transformed as before.

In this thesis, feature engineering using fastKNN was implemented using the *PfastKNN()* and *PfastKNNset()* functions which are included in the appendix. The *PfastKNNset()* function performs LOOCV for each new training set to avoid overfitting, and the *PfastKNN()* function computes the K new features for each group in the training set as explained in Section 2.3.1. The value of K used in the fastKNN feature extraction procedure is the same as the value of K used in the KNN classifier. Feature extraction using fastKNN with $K = 5$ is also implemented in the SVM classification procedure in the five binary data sets (therefore, the RBF SVM is applied to 10 (5×2) new features extracted from fastKNN).

The misclassification test error rate for the KNN and fastKNN procedures were calculated on each split of the data set.

5.5.2 LDA and DLDA

The LDA classifier was only applied to reduced versions of the colon cancer, leukemia, Sim1 and Sim3 data sets using the *PLDA()* programme as in the steps explained in Section 2.4. The *PLDA()* programme returns the classification for each observation in the test data set and this programme is included in Appendix B. Note that the data sets first underwent VS using the NSC procedure before applying LDA. This was done to overcome the singularity problem arising when LDA is applied to data sets where $p > N$.

The DLDA classifier was also only applied to the binary practical data sets using the *PdiagLDA()* programme which is included in Appendix B. The *PdiagLDA()* programme performs the necessary steps to calculate the DLDA discriminant score for the two populations in the test data as in Equation (2.3) and it returns the classification predictions for the test cases.

Note that the data sets were not standardised before applying the LDA and DLDA classifiers.

5.5.3 NSC

The NSC variable selection and classification procedures were applied to both the binary and multi-class data sets. The *pamr.train()* function from the *pamr* package in R was applied to the training data set to obtain a vector of 30 candidate values for the threshold Δ. In an attempt to reduce the computations in determining the optimal Δ value in the NSC procedure, the smallest seven and largest eight of these threshold values from the 30 candidate values obtained in the *pamr* package were omitted and only the remaining 15 candidate values were considered. The NSC procedure is computationally extremely expensive and it was initially thought that an even smaller range of candidate Δ values, namely ten, should be considered to speed up the process of finding an optimal value of Δ. However, data exploration highlighted the fact that the results of the NSC-KNN and NSC-SVM procedures are both quite sensitive to the choice of Δ, and therefore it can be concluded that it was better to use 15 candidate values from the default range of 30 possibilities for the CV choice of Δ. The optimal Δ value was then determined using LOOCV on the training set, by fitting the necessary classifier to the selected genes for each of the 15 Δ candidate values and reporting the training misclassification error rate.

The optimal Δ value was the maximum Δ candidate value (smallest number of selected genes) which achieves the lowest training error rate.

The NSC programme in R (which can be found in Appendix B) was thereafter applied to the training data using the optimal Δ value to obtain the indices of the genes which play a role in classification, disregarding the rest.

The NSC classification procedures were applied using the NSC discriminant score for the classes as in Equation (2.3) in Section 2.5, using the programme *Pnsc_class()* for the binary data sets and *Pnsc_class_mult()* for the SRBCT multi-class data set. Both of these programmes are included in Appendix B.

5.5.4 SVM

The RBF SVM classifier was only applied to the binary data sets, and not to the multi-class data set. The *Psvm* programme included in Appendix B was used for this purpose.

The first step in the *Psvm* programme determines the optimal γ value in the RBF kernel on each split of the training data using the *sigest* function in the *kernlab* package in R. The next step in the *Psvm* programme is to apply the *ksvm()* function from the *kernlab* package in R to the data set using the following arguments. The argument *type="C-svc"* is given to indicate that the SVM is being used in a classification setting. In order to indicate that the Radial Basis Function kernel ("Gaussian" function) must be used to compute inner products in the feature space, *kernel="rbfdot"* is used. The argument *kpar=list(sigma=opt.gam)* is used to set the single hyper-parameter in the kernel RBF function, the γ value, equal to the optimal γ value obtained from the data set in Step 1. As mentioned in Chapter 2, for the purpose of this thesis, the C parameter was set to 1, 1 000 and 10 000 in order to examine the possible influence of this parameter on the classification performance of the SVM. The kernel matrix in Equation (2.13) was computed using the *kernelMatrix()* function.

Finally, the *Psvm* programme outputs the real $\hat{f}(\boldsymbol{x})$ values in the SVM classifier, as calculated in Equation (2.13).

5.6 CROSS-VALIDATON

James *et al.* (2013:181) describe cross-validation or Lachenbruch's holdout procedure as one of the most popular (and simplest) methods for estimating the prediction error and thereby assessing the performance of a classification method. Leave-one-out CV omits one observation at a time, which is referred to as the “holdout” observation, from the data set. The remaining observations can be regarded as the training data set and are used to create a classification function (Johnson and Wichern, 2007:599). The classification function is then used to classify the “holdout” observation, which can be regarded as the test data. The cross-validation procedure starts with leaving out the first observation and is repeated N times until every observation has been left out and classified using the classification function obtained without that specific case. James *et al.* (2013:184) state that in classification settings the LOOCV error rate is calculated as:

$$CV_{(N)} = \frac{1}{N}\sum_{i=1}^{N} Err_i. \tag{5.4}$$

In (5.4), $Err_i = I(y_i \neq \hat{y}_i)$, where I is the indicator function which equals 0 if the i^{th} observation is correctly classified $(y_i = \hat{y}_i)$, and 1 if it is misclassified $(y_i \neq \hat{y}_i)$. This procedure usually provides a good estimate of the actual error rate for each classification method, and will be used to determine the optimal values of the tuning parameters that will be used in the classification and variable selection procedures considered in the thesis.

K-fold cross-validation is used in extremely large data sets, and occurs when the holdout procedure is conducted on K folds of observations as opposed to individual observations (James *et al.*, 2013:181). The data set is randomly divided into K folds or parts. The fold sizes are calculated by dividing the number of data cases by K, the number of folds. The folds will be approximately or exactly the same size, depending on whether or not the number of data cases is divisible by K. The test data (validation group) is left out and a classification function is obtained on the data of the remaining $K-1$ parts. The fitted classification function is then used to classify each observation in the test part and the CV error for the k^{th} held-out fold is calculated. This is repeated until each of the K folds have been left out.

The misclassification rate for the classification method using different tuning parameter values (thresholding values) is estimated using the average of the misclassification rates obtained from the different held-out parts of the data set. The performance of the classification methods using a range of candidate threshold values is examined on the data sets and evaluated in the following section using their respective cross-validation error rates corresponding to the tuning parameters.

Data preparation for implementing the different thresholding values in the classification procedures on all six data sets was done in the following way. Firstly, the data set was randomly split into a training data set (80%) and a test data set (20%). Care was taken to ensure that the populations were represented in the training and the test data in the same proportion as in the original full data set. The whole process was repeated 100 times (therefore, 100 different random splits into training and test cases were performed), and the results were averaged.

The KNN classifier was applied using K equal to 1, 3 and 5 and the CV classification error rate was calculated on each split for every candidate threshold value. The SVM was applied using C equal to 1,1 000 and 10 000 and the CV classification error rate was calculated on each split for every candidate threshold value. The optimal threshold value for the first split was determined as the threshold value corresponding to the lowest CV error rate. This is done for all 100 splits and the frequencies with which the candidate threshold values are selected as optimal, are recorded.

Therefore, in this thesis, optimal threshold values for the following tuning parameters are determined using CV and are then kept fixed throughout the remainder of the analysis. The three tuning parameters are: the correlation threshold value, the two-sample t-test p-value threshold and the shrinkage value Δ in the NSC procedure.

The candidate values for the two-sample t-test are pre-determined while these candidate values are computed from the data in the case of correlation and NSC thresholding.

The following tables report the selection frequency of the candidate correlation, p-value in the two-sample t-test and NSC parameter Δ threshold values. These frequencies are obtained by implementing KNN classification for different values of K, SVM classification using different values of the C parameter and finally the NSC classification procedures applied to the six high-dimensional microarray data sets. Note that the SVM classifier is not applied to the SRBCT multi-class data set.

5.6.1 Determining the optimal correlation threshold value

Five candidate correlation threshold values were considered for each of the binary data sets. These five candidate values were calculated for each of the data sets using the following procedure: Calculate the absolute correlation of each predictor variable with the response and determine the maximum absolute correlation value, cor_{max}. Then the five candidate correlation threshold values range in equal increments from 0.1 to 0.8$\times$ cor_{max}.

Note that the highest candidate correlation threshold value considered is 80% of cor_{max} to ensure that the selected threshold is not too large, thereby ensuring that at least one feature is later on extracted after the data has been split into training and test parts. Finally, although the range of candidate correlation thresholds remains constant for each data set, LOOCV is used to determine optimal correlation threshold on each split, and therefore the optimal correlation threshold value used in feature extraction typically varied from split to split.

Selecting a large threshold for the absolute correlation between the response and a predictor variable implies that only the genes that are highly correlated with Y remain in the model and a large reduction of approximately uncorrelated, hopefully irrelevant genes occurs. Conversely, selecting a smaller correlation threshold value implies that there will be a smaller reduction in the number of genes. This will be appropriate if a large subset of genes is required to achieve accurate classification using the KNN and RBF SVM classifiers.

The table below summarises the frequencies with which the five candidate correlation threshold values determined on the colon cancer data, were selected over the 100 random splits. The frequencies are reported when $K \in \{1, 3, 5\}$ is used in the KNN classifier and when $C \in \{1, 1\,000, 10\,000\}$ is used in the RBF SVM classifier.

Table 5.3: LOOCV correlation threshold frequencies using KNN & SVMs

Correlation threshold value	**Frequency**					
	KNN Classifier			**SVM Classifier**		
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C = 1\,000$	$C = 10\,000$
0.1000	25	24	17	4	33	35
0.2013	32	35	47	18	43	40
0.3026	7	12	7	31	11	11
0.4039	9	10	21	23	9	10
0.5053	27	19	8	24	4	4
Total	100	100	100	100	100	100

Before analysing the results reported in Table 5.3 it should be noted that in the colon data set gene 249 has the maximum absolute correlation with Y at a value of 0.6318, while gene 1 122 has the minimum absolute correlation value of 0.0001 with Y. Additionally, in the full colon data set the median absolute correlation with Y is 0.1272 and 55 genes have an absolute correlation value greater than 0.5053 with Y.

The results reported in Table 5.3 indicate that for the KNN classifier with $K = 1$, 3 and 5 the optimal correlation threshold value on the colon data set is 0.2013 as it has the maximum selection frequency over the 100 random splits. Therefore, based on the 100 random splits of the colon data set, it appears that the subset of genes which have an absolute correlation above 0.2013 with Y will achieve the maximum classification accuracy for $K \in \{1, 3, 5\}$ in the KNN classifier.

Additionally, Table 5.3 suggests that for the colon data set the optimal correlation threshold value using $C = 1$ in the SVM classifier is 0.3026, while for $C = 1\ 000$ and 10 000 the optimal correlation value is 0.2013.

The results summarised in Table 5.3 indicate that the optimal correlation threshold value is selected between 0.1000 and 0.2013 majority of the time for large C parameter values (76% for $C =$1 000 and 75% for $C =$10 000). However, for $C = 1$ in the SVM classifier the optimal correlation threshold value is selected between 0.3026 and 0.5053, 78% of the time. Therefore, it is clear from the results given in Table 5.3 that for a small C value ($C = 1$) the RBF SVM classifier performs better on a highly-reduced subset of the genes by selecting a high correlation threshold. This is what is expected since a small C value in the SVM classifier implies that the variables are not being penalised heavily. Therefore, due to the light regularisation a larger (stricter) correlation threshold value should be implemented to ensure that fewer variables remain in the model. Conversely, the smaller correlation threshold value of 0.2013 selected for the two larger values of C suggests that the RBF SVM classifier is more accurate on a larger subset of the original genes in the colon data set. This is what is intuitively expected, since a larger C parameter value value implies more regularisation and therefore the SVM classifier can afford to include more variables and thus a high frequency of smaller correlation threshold values is expected.

The table below summarises the frequencies with which the five candidate correlation threshold values determined on the leukemia data set, were selected over the 100 random splits. These frequencies are reported for $K \in \{1, 3, 5\}$ in the KNN classifier and $C \in \{1, 1\ 000, 10\ 000\}$ in the RBF SVM classifier.

Before analysing the results reported in Table 5.4, it should be noted that in the leukemia data set gene 1 182 has the maximum absolute correlation with Y at a value of 0.8597, while gene 277 is the least correlated with Y. The median absolute correlation value in the leukemia data set is 0.1874, and 1 024 genes have a correlation with Y that is smaller than the minimum threshold in Table 5.4. Additionally, 28 genes in the leukemia data set have an absolute correlation with Y that is greater than the maximum threshold value (0.6878) in Table 5.4.

Table 5.4: LOOCV correlation threshold frequencies using KNN & SVMs

Correlation threshold value	**Frequency**					
	KNN Classifier			**SVM Classifier**		
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C = 1\ 000$	$C = 10\ 000$
0.1000	33	42	14	96	97	97
0.2469	30	19	44	2	2	2
0.3939	29	22	27	2	1	1
0.5408	5	16	14	0	0	0
0.6878	3	1	1	0	0	0
Total	100	100	100	100	100	100

The results reported in Table 5.4 suggest that the optimal correlation threshold value is 0.1000 for the 1-NN and 3-NN classifiers on the leukemia data set, while, for the 5-NN classifier the optimal correlation threshold value is 0.2469. The results given in Table 5.4 imply that the optimal correlation threshold value for the SVM classifier on the leukemia data set is 0.1000 for $C \in \{1, 1\ 000, 10\ 000\}$, since it is selected at least 96% of the time over the 100 random splits. Unlike for the colon data set, the results for the SVM classifier applied to the leukemia data set are robust with respect to C and always select the correlation threshold value of 0.1000.

Therefore, from the results summarised in Table 5.4 it appears that a larger subset of genes in the leukemia set (excluding only the genes that have a very small absolute correlation less than 0.1000 with Y) achieves maximum classification accuracy using the KNN and SVM classifiers.

The three tables below summarise the frequencies with which the five candidate correlation threshold values were selected for the 100 random splits of the first, second and third simulated data sets respectively. These are reported for $K \in \{1, 3, 5\}$ in the KNN classifier and $C \in \{1, 1\ 000, 10\ 000\}$ in the RBF SVM classifier.

The results for the first simulated data set are:

Table 5.5: LOOCV correlation threshold frequencies for Sim1 using KNN & SVMs

Correlation threshold value	**Frequency**					
	KNN Classifier			**SVM Classifier**		
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C = 1\,000$	$C = 10\,000$
0.1000	5	10	15	2	18	18
0.1850	13	7	18	26	15	14
0.2700	13	15	13	14	12	13
0.3550	46	46	36	47	45	45
0.4399	23	22	18	11	10	10
Total	100	100	100	100	100	100

It is evident from the results in Table 5.5 that the optimal correlation threshold value is 0.3350 for both the KNN and SVM classifier. Note that 0.3550 is the second largest threshold value tested in Table 5.5, which suggests that there is a substantial number of irrelevant genes that should be removed by correlation thresholding to achieve maximum classification accuracy over the 100 random splits of Sim1. The results for the SVM classifier on Sim1 do not follow what is intuitively expected and the optimal correlation threshold is robust to the value of C. Additional investigation of the Sim1 data set showed that 54 genes had an absolute correlation with Y above 0.3350 and will be retained in the model when implementing correlation thresholding at the value of 0.3350. Since it is known that for the Sim1 data set only 10% (100) of the genes are relevant in distinguishing between P_1 and P_2, the results in Table 5.5 confirm that a large reduction in the number of genes retained in the model is necessary.

Table 5.6 summarises the results for the second simulated data set. Note that additional computations showed that only six genes (the genes corresponding to the indices 382, 415, 447, 471, 473 and 408) have an absolute correlation above 0.4296 with the response.

It is clear from Table 5.6 that the optimal correlation thresholding value on the second simulated data set is 0.4296 for all values of K and C in the KNN and RBF SVM classifiers respectively. Furthermore, it should be noted that unlike in Table 5.5 the optimal correlation threshold value was selected for the majority of the 100 random splits.

Table 5.6: LOOCV correlation threshold frequencies for Sim2 using KNN & SVMs

Correlation threshold value	**Frequency**					
	KNN Classifier			**SVM Classifier**		
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C = 1\,000$	$C = 10\,000$
0.1000	19	14	12	0	4	4
0.1824	4	3	5	4	6	6
0.2648	1	2	0	4	1	1
0.3472	11	11	12	15	11	10
0.4296	65	70	71	77	78	79
Total	100	100	100	100	100	100

The selection of the large optimal correlation threshold value suggests that only the six genes that have a high correlation of above 0.4296 with Y are significant and thus there will be a substantial reduction in the number of genes retained in the model for the Sim2 data set.

This can be explained by the fact that the correlation coefficient cannot be expected to accurately identify the important variables in cases where the two populations differ with respect to scale. Note that the six genes referred to above actually do not distinguish between the two populations in the Sim2 data set and are therefore most probably identified by chance as being important. One would expected that any decent thresholding method would select more than six significant genes in a simulated data set that is known to have 100 relevant genes.

The results for the third simulated data set are as follows:

Table 5.7: LOOCV correlation threshold frequencies for Sim3 using KNN & SVMs

Correlation threshold value	**Frequency**					
	KNN Classifier			**SVM Classifier**		
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C = 1\,000$	$C = 10\,000$
0.1000	10	21	38	30	44	45
0.1948	22	12	13	13	11	11
0.2897	25	21	12	15	8	7
0.3845	31	36	26	26	20	21
0.4793	12	10	11	16	17	16
Total	100	100	100	100	100	100

Table 5.7 indicates that for Sim3 the optimal correlation threshold value is dependent on the value of K in the KNN classifier. For the small values of K (1 and 3) the optimal correlation threshold value is 0.3845, implying that when only a few nearest neighbours are considered a large reduction in the number of genes is necessary to achieve good classification accuracy (computations show that 44 genes in the Sim3 data set have a correlation above 0.4793 with Y). For the 5-NN classifier a larger subset of approximately 624 genes is required since the optimal correlation threshold value is 0.1000, which is the smallest of the five candidate threshold values considered in Table 5.7.

Using the SVM classifier with $C \in \{1, 1\,000, 10\,000\}$, it is clear from Table 5.7 that 0.1000 is the optimal correlation threshold value for Sim3.

5.6.2 Determining the optimal two-sample t-test p-value threshold

Tables 5.8 through 5.12 report the selection frequencies of four pre-specified p-values in the two-sample t-test (namely 0.001, 0.01, 0.05 and 0.10) over the 100 random splits of the binary data sets. These frequencies are reported for $K \in \{1, 3, 5\}$ in the KNN classifier and for $C \in \{1, 1\,000, 10\,000\}$ in the RBF SVM classifier.

Selecting a small p-value threshold, for example 0.001, in the two-sample t-test procedure suggests that the VS criterion is very strict, since only the genes with a p-value less than 0.001 are considered significant and selected for inclusion in the model. Note that the smaller the pre-specified p-value threshold, the greater the difference between the two sample means for a specific gene must be for the gene to be selected as significant and to remain in the model. Therefore, the smaller the p-value threshold used in the two-sample t-test thresholding, the stricter the variable selection criterion and the fewer genes will remain in the model. Conversely, for a larger p-value threshold used in the two-sample t-test, the variable selection criterion is less strict and fewer genes will be eliminated from the model.

Table 5.8 below summarises the frequencies with which the four pre-specified p-value thresholds in the two-sample t-test selection procedure were selected in 100 random splits of the colon cancer data set.

Before analysing the results reported in Table 5.8 it should be noted that in the colon data set, gene 249 had the minimum calculated p-value < 0.0001 and is considered the most significant gene in terms of the two-sample t-test. Note gene 249 was also ranked the most important using correlation thresholding. Additionally, further analyses indicated that for the 2 000 genes in the colon data set the median p-value is 0.1623 and that 1 215 genes are not selected at a pre-specified p-value $=$ 0.100.

Table 5.8: LOOCV t-test p-value frequencies using the KNN & SVM classifier

p-value	Frequency					
	KNN Classifier			SVM Classifier		
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C = 1\ 000$	$C = 10\ 000$
0.001	31	32	52	67	17	19
0.010	18	18	12	20	31	32
0.050	20	27	24	9	22	29
0.100	31	23	12	4	30	20
Total	100	100	100	100	100	100

It is evident from Table 5.8 that for all three values of K in the KNN classifier applied to the colon data set the optimal p-value threshold in the two-sample t-test is 0.001. For the 1-NN classifier in Table 5.8, the p-value threshold equal to both 0.001 and 0.100 is selected in 31% of the 100 random splits; however the smaller of the two p-values is used in further analyses as it results in more genes being eliminated from the model. Considering Table 5.8, the optimal p-value is 0.001 in the two-sample t-test if the SVM classifier with C =1 is applied to the colon data set. However, for C = 1 000 and 10 000 in the SVM classifier, the optimal p-value threshold in the two-sample t-test is 0.01. The small optimal p-value threshold for both the KNN and SVM classifiers suggests that many of the genes retained in the colon data set differ significantly in the tumour and normal population groups.

The table below summarises the frequencies with which the four pre-specified p-value thresholds in the two-sample t-test selection procedure were selected in 100 random splits of the leukemia data set. In the leukemia data set, gene 436 gave the minimum p-value of approximately 0 (indicating that it is the most significant gene in distinguishing between AML and ALL leukemia patients), while gene 277 has the maximum p-value of 0.4969. The median p-value in the two-sample t-tests is 0.0575 and 704 genes have a p-value $\leq$ 0.001 while 1 479 genes have a p-value $\geq$ 0.1.

From Table 5.9 it can be observed that for all three values of K in the KNN classifier and all three values of the C parameter in the SVM classifier, the optimal p-value in the two-sample t-test VS procedure applied to the leukemia cancer data set is 0.001.

Table 5.9: LOOCV t-*test* p-value frequencies using the KNN & SVM classifier

p-value	Frequency					
	KNN Classifier			SVM Classifier		
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C = 1\ 000$	$C = 10\ 000$
0.001	72	94	87	95	92	92
0.010	4	3	11	4	6	6
0.050	14	2	1	1	1	1
0.100	10	1	1	0	1	1
Total	100	100	100	100	100	100

Note that additional computations showed that 704 genes (19.71% of the total genes) in the leukemia data set yielded a p-value less than 0.001. As in Table 5.8, the small optimal p-value threshold for both the KNN and SVM classifiers suggests that only a small subset of genes that differ significantly between the AML and ALL groups are required to achieve the optimal classification accuracy.

The following three tables summarise the frequencies with which the four pre-specified p-value thresholds in the two-sample t-test VS procedure were selected in 100 splits of the first, second and third simulated data sets respectively. Table 5.10 provides the results for the first simulated data set.

Table 5.10: LOOCV t-test p-value frequencies for Sim1 using KNN & SVMs

p-value	Frequency					
	KNN Classifier			SVM Classifier		
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C = 1\ 000$	$C = 10\ 000$
0.001	13	7	9	5	6	6
0.010	49	56	55	36	37	37
0.050	26	26	21	39	39	39
0.100	12	11	15	20	18	18
Total	100	100	100	100	100	100

From Table 5.10 it is evident that the optimal p-value threshold used in the two-sample t-test on the Sim1 data set is 0.010 for all values of K in the KNN classifier. However, for all three values considered for the C parameter in the SVM classifier the optimal p-value threshold used in the two-sample t-test is 0.050, although it was only selected 5.13% and 7.69% more times than the threshold of 0.01.

Note that in the Sim1 data set, 46 genes have a calculated p-value less than 0.010 and 143 genes have calculated p-values less than 0.050. In summary, Table 5.10 indicates that the optimal p-value used in the two-sample t-test thresholding for the KNN and SVM classifiers is selected as 0.010 or 0.050 for at least 75% of the random splits.

Table 5.11 provides the results for the second simulated data set.

Table 5.11: LOOCV t-test p-value frequencies for Sim2 using KNN & SVMs

p-value	Frequency					
	KNN Classifier			SVM Classifier		
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C = 1\,000$	$C = 10\,000$
0.001	76	70	71	56	62	61
0.010	23	29	29	43	38	39
0.050	1	1	0	1	0	0
0.100	0	0	0	0	0	0
Total	100	100	100	100	100	100

From Table 5.11 it is clear that the optimal p-value threshold in the two-sample t-test thresholding on the second simulated data set is 0.001 for all values of K in the KNN classifier as well as all three values considered for the C parameter in the SVM classifier. Note that only three genes (genes with indices 382, 471 and 908) in the Sim2 data set have a p-value less than 0.001 and would be retained in the model if two-sample t-test thresholding is implemented with a threshold of 0.001.

Table 5.12 provides the results for the third simulated data set.

Table 5.12: LOOCV t-test p-value frequencies for Sim3 using KNN & SVMs

p-value	Frequency					
	KNN Classifier			SVM Classifier		
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C = 1\,000$	$C = 10\,000$
0.001	16	19	18	25	20	20
0.010	35	43	45	43	42	40
0.050	32	26	28	19	21	22
0.100	17	12	9	13	17	18
Total	100	100	100	100	100	100

It is evident from the results summarised in Table 5.12 that the optimal p-value threshold for applying two-sample t-test thresholding on the third simulated data set is 0.01 for the three values considered for K in the KNN classifier and for the three values of the C parameter in the RBF SVM classifier. Extra calculations showed that 50 genes in Sim3 have p-values less than 0.01.

5.6.3 Determining the optimal NSC Δ parameter value

As explained earlier, for every data set analysed in this thesis, 15 candidate NSC Δ parameter threshold values were considered. In each case these values were obtained from the *pamr* package in R. In addition, in scenarios where there was more than one Δ value yielding the smallest error rate, the larger Δ threshold value was selected for use in further analyses since it selects a smaller subset of genes than the larger threshold values. Note that in the algorithm for determining the Δ_{opt} threshold value, the selection of genes for a given value of Δ is obtained separately for each of the LOOCV trials to avoid selection bias and unrealistically optimistic CV error rates.

The 15 different subsets of genes corresponding to the 15 candidate Δ values in the NSC procedure are identified for each random split of the data. These subsets are then used in KNN classification with $K \in \{1, 3, 5\}$, SVM classification with $C \in \{1, 1\,000, 10\,000\}$ and the NSC classifier. The frequencies with which each candidate value leads to a best subset are recorded. Note that due to computational considerations only a single optimal value determined using the NSC classifier will be used in the remainder of the analyses.

The results pertaining to the colon cancer data set are given in Table 5.13 with the results for the leukemia data set provided in Table 5.14.

The results summarised in Table 5.13 indicate that the optimal NSC Δ shrinkage value varies for different values of K used in the KNN classifier. The $\Delta_{opt} = 2.4612$ for $K = 1$, while for $K \in \{3, 5\}$ the optimal value for the shrinkage parameter is $\Delta_{opt} = 3.9757$.

Interestingly, the results summarised in Table 5.13 indicate that for the 3-NN and 5-NN classifiers the optimal NSC value is selected a majority of the time. However, the results reported for the 1-NN classifier in Table 5.13 are dispersed over the range of the 15 candidate Δ values. This could suggest that the Δ_{opt} value is sensitive to the training-test split in the 1-NN classifier.

Table 5.13: NSC Δ LOOCV frequencies using the KNN, SVM & NSC classifiers

NSC Δ parameter	Frequency						
	KNN Classifier			SVM Classifier			NSC Classifier
	$K = 1$	$K = 3$	$K = 5$	$C = 1$	$C =$ 1 000	$C =$ 10 000	
1.3252	2	9	6	5	8	11	4
1.5146	7	2	1	5	7	9	10
1.7039	2	5	1	5	3	5	17
1.8932	3	6	2	4	2	5	17
2.0825	10	4	2	1	4	5	15
2.2718	14	1	1	1	4	4	14
2.4612	16	5	1	3	1	3	9
2.6505	9	3	2	4	4	3	9
2.8398	8	1	0	3	0	2	3
3.0291	5	1	1	1	2	0	2
3.2184	8	3	2	0	6	1	0
3.4078	0	4	1	3	1	2	0
3.5971	4	1	2	2	3	9	0
3.7864	0	3	2	1	20	12	0
3.9757	12	52	76	62	35	29	0
Total	100	100	100	100	100	100	100

In Table 5.13 the results indicate that the NSC Δ_{opt} value is 3.9757 for all three values of the C parameter used in the RBF SVM classifier on the colon cancer data set. For $C = 1$ the Δ_{opt} value is selected a majority of the time, while for larger values of the C parameter, $C \in \{1\ 000, 10\ 000\}$, the selection frequencies of the Δ value are more spread out over the range of 15 candidate values.

It is clear from the last column in Table 5.13 that the Δ_{opt} shrinkage parameter in the NSC classifier is both 1.7039 and 1.8932. However, as explained previously, $\Delta = 1.8932$ is selected as the optimal value for the NSC classifier since it is the larger of the two values resulting in a smaller subset of genes eventually being selected. Different from the results for the KNN and SVM classifiers in Table 5.13, the majority of the selection frequencies for the NSC classifier is centred in an approximately bell-shape around the optimal value.

Due to computational considerations only the Δ_{opt} value of 1.8932 determined using the NSC classifier will be used in the colon data set from here on out.

The table below summarises the results for the leukemia data set.

Table 5.14: NSC Δ LOOCV frequencies using the KNN, SVM & NSC classifiers

NSC Δ parameter	Frequency						
	KNN Classifier			SVM Classifier			NSC Classifier
	$K = 1$	$K = 3$	$K = 5$	C =1	C =1000	C =10 000	
1.1629	2	2	0	0	0	0	0
1.3290	3	2	3	3	4	3	1
1.4952	5	11	0	6	5	6	11
1.6613	9	6	6	7	7	5	19
1.8274	8	8	11	13	19	18	18
1.9936	2	8	9	11	12	13	16
2.1597	3	6	8	4	5	5	28
2.3258	5	5	8	4	3	3	6
2.4920	10	12	12	9	4	2	1
2.6581	11	8	6	6	11	14	0
2.8242	9	3	6	6	4	8	0
2.9904	21	9	15	17	15	12	0
3.1565	6	9	5	3	2	3	0
3.3226	2	1	1	1	1	2	0
3.4888	4	10	10	10	8	6	0
Total	100	100	100	100	100	100	100

Table 5.14 shows that the Δ_{opt} value varies for $K \in$ {1, 3, 5} in the KNN classifier. The $\Delta_{opt} =$ 2.9904 for $K =$ 1 and 5, while for the 3-NN classifier $\Delta_{opt} =$ 2.4920. Unlike in Table 5.4 and Table 5.9 the results are spread out over the candidate Δ values which could suggest that Δ_{opt} is sensitive to the training-test split as well as to the value of K in the KNN classifier. Additionally, the results in Table 5.14 suggest that Δ_{opt} is sensitive to the C parameter value used in the SVM classifier. As seen in Table 5.14, for a small value of C in the SVM classifier, C =1, the Δ_{opt} = 2.9904. However, for the two large C parameter values in the SVM classifier, *i.e.* $C \in$ {1 000, 10 000}, Δ_{opt} is much smaller at 1.8274. Therefore, for small values of C in the SVM a higher Δ value is required to obtain classification accuracy resulting in more shrinkage in the leukemia data set and therefore a smaller subset of genes remaining in the model. This is expected as a small value of C leads to a smooth, potentially underfit boundary due to a small amount of regularisation.

In Table 5.14 it is clear for the NSC classifier $\Delta_{opt} = 2.1597$. Unlike in the KNN and SVM classifiers in Table 5.14, the majority of the Δ selection frequencies for the NSC classifier is centred in an approximately bell-shape around the Δ_{opt} value.

Note that in the paper by Tibshirani *et al.* (2002:6570), the authors state that the minimum CV error occurs around $\Delta = 1.4$, resulting in 1 000 genes remaining in the model. The authors therefore used $\Delta = 4.06$ to select a more manageable set of only 21 genes. Applying this to the range of 30 candidate Δ values computed for the leukemia data set and considered in this thesis, $\Delta = 1.4$ falls between the 7th (0.9968) and the 8th (1.1629) value and $\Delta = 4.06$ falls between the 25th (3.9871) and 26th (4.1533) value computed for the leukemia data set.

Tables 5.15 through 5.17 provide the frequencies of each of the 15 candidate Δ parameter threshold values in the NSC VS procedure using the NSC classifier pertaining to 100 random splits in the three simulated data sets. In order to reduce computations only the NSC classifier is applied to the reduced data sets using the NSC VS procedure.

Table 5.15: NSC Δ LOOCV frequencies for Sim1 using the NSC classifier

NSC Δ parameter	**Frequency**
0.5013	36
0.5729	30
0.6446	18
0.7162	9
0.7878	3
0.8594	1
0.9310	2
1.0026	0
1.0743	1
1.1459	0
1.2175	0
1.2891	0
1.3607	0
1.4323	0
1.5040	0
Total	100

In Table 5.15 it is evident that the optimal NSC parameter value for the NSC classifier on the first simulated data set is $\Delta_{opt} = 0.5013$ as it has the maximum frequency of 36 out of the 100 random splits. Table 5.15 shows that the selection frequencies of the 15 candidate NSC parameter values decrease as Δ increases in size. Therefore, a smaller Δ value and subsequently a smaller reduction in the number of genes used, results in a lower LOOCV error when implementing the NSC classifier on 100 random splits of the Sim1 data set.

From the results given in Table 5.15 one should note that the $\Delta_{opt} = 0.5013$ and that in the first simulated data set the first $p_{rel} = 100$ of the $p = 1\ 000$ genes are differentially expressed in the two classes by a constant amount of 0.5 as there is a location difference.

The results for the second simulated data set are given below.

Table 5.16: NSC Δ LOOCV frequencies for Sim2 using the NSC classifier

NSC Δ parameter	Frequency
0.4729	0
0.5405	3
0.6080	7
0.6756	15
0.7432	12
0.8107	12
0.8783	11
0.9458	17
1.0134	12
1.0809	10
1.1485	1
1.2161	0
1.2836	0
1.3512	0
1.4187	0
Total	100

In Table 5.16, the optimal NSC procedure value using the NSC classifier is $\Delta_{opt} = 0.9458$ with a maximum frequency of 17 out of the 100 repetitions.

Table 5.17 below summarises the results for the third simulated data set.

Table 5.17: NSC Δ LOOCV frequencies for Sim3 using the NSC classifier

NSC Δ parameter	Frequency
0.5109	7
0.5838	14
0.6568	22
0.7298	26
0.8028	6
0.8758	6
0.9487	7
1.0217	2
1.0947	6
1.1677	4
1.2407	0
1.3136	0
1.3866	0
1.4596	0
1.5326	0
Total	100

It is clear from the results reported in Table 5.17 that $\Delta_{opt} = 0.7298$ as it has the maximum selection frequency of 26 out of the 100 random splits. Note that the Δ_{opt} value in Table 5.17 is larger than the Δ_{opt} value of 0.5013 selected for the Sim1 data set. However, the Sim3 data set has both a location and scale change of 0.5 applied to the first p_{rel} genes in P_2 and therefore it is expected that a higher Δ_{opt} value is selected for the Sim3 data set in Table 5.17 than for the Sim1 data set with only a location difference in Table 5.15.

5.6.4 Determining the optimal threshold values for the SRBCT data set

As in the binary classification applications explained in Section 5.6.1, five candidate correlation thresholds were considered for the SRBCT data set by calculating the absolute correlation of each predictor variable with the response and determining the maximum absolute correlation value, cor_{max}. Then the five candidate correlation threshold values range in equal increments from 0.1 to 0.9$\times$ cor_{max}. Note that the highest correlation threshold value considered was 90% of the cor_{max} to ensure that at least one gene was extracted using correlation thresholding. Finally, although the range of candidate correlation thresholds remains constant for the SRBCT data set, LOOCV was used to determine the optimal correlation threshold on each split, and this optimal correlation threshold value used in feature extraction could vary from split to split.

The table below summarises the selection frequencies of the five candidate correlation threshold values for the 100 random splits of the SRBCT data set. This is reported for $K \in \{1, 3, 5\}$ in the KNN classifier.

Table 5.18: LOOCV correlation threshold frequencies using the KNN classifier

Correlation threshold value	**Frequency**		
	$K = 1$	$K = 3$	$K = 5$
0.1000	0	1	0
0.2563	0	0	0
0.4126	0	0	0
0.5690	0	0	0
0.7253	100	99	100
Total	100	100	100

The results given in Table 5.18 indicate that for the SRBCT data set, the optimal correlation threshold value is independent of the value of K in the KNN classifier and is 0.7253 for all values of K. Therefore, from the results in Table 5.18 it can be concluded that only the genes that have an absolute correlation greater that 0.7253 with Y in the SRBCT data set should be retained in the model. Note that gene 1 194 has the maximum absolute correlation of 0.8059 with Y out of all of the 2 308 genes in the SRBCT data set. Furthermore, only five genes in the SRBCT data set have a correlation with Y above the maximum threshold value of 0.7253 in Table 5.18. These are the genes with indices 187, 509, 1 003, 1 1194 and 2 046.

Finally, one should take note that the results reported in Table 5.18 are less dispersed over the range of candidate absolute correlation values than in both of the binary practical data sets reported in Tables 5.3 through 5.7.

Table 5.19 reports the frequencies for the 15 candidate Δ parameter values in the NSC extraction procedure for 100 random splits of the SRBCT data set for $K \in \{1, 3, 5\}$ in the KNN classifier as well as these frequencies for the NSC classifier.

Table 5.19: LOOCV frequencies for the NSC Δ threshold value using KNN & NSC

NSC Δ parameter	Frequency			
	KNN Classifier			NSC Classifier
	$K = 1$	$K = 3$	$K = 5$	
1.9430	0	0	0	0
2.2206	0	0	0	0
2.4982	0	0	0	0
2.7757	0	0	0	0
3.0533	0	0	0	23
3.3309	0	0	0	41
3.6085	0	0	0	31
3.8860	0	0	0	5
4.1636	0	0	0	0
4.4412	0	0	0	0
4.7188	1	0	0	0
4.9963	4	0	2	0
5.2739	12	18	19	0
5.5515	8	25	38	0
5.8290	75	57	41	0
Total	100	100	100	100

As shown in Table 5.19, $\Delta_{opt} = 5.8290$ for $K \in \{1, 3, 5\}$ in the KNN classifier on the SRBCT data set. However, for the NSC classifier $\Delta_{opt} = 3.3309$. It is interesting to note that for the KNN classifier in Table 5.19, only the five maximum threshold values are selected over the 100 random splits, with the two largest Δ values in Table 5.19 being selected the majority of the time. For the NSC classifier the selected Δ frequencies are spread over only four candidate values ranging from 3.0533 to 3.8860, and all values falling outside this range are never selected in the 100 random splits of the SRBCT data set.

To ensure comparability across classifiers going forward, $\Delta_{opt} = 3.3309$ will be implemented in the NSC variable selection procedure applied to the SRBCT data set.

Note that Tibshirani *et al.* (2001) report that using 10-fold cross-validation on the SRBCT data set, ensuring that the classes were distributed proportionally among each of the 10 parts, yielded that the optimal value for the NSC parameter is $\Delta = 4.34$, which selects 43 active genes.

5.7 SUMMARY

This chapter started with a section in which a detailed description of DNA microarray data sets was provided. This was followed by a description of the three practical data sets and three simulated data sets considered in the thesis. The chapter then provided a summary and overview of the classification procedures that were implemented in the analysis of the binary and multi-class data sets. This lead to a description of how the base classifiers that were used in the empirical study were implemented in R.

The chapter then introduced and briefly discussed the concept of LOOCV and its use to determine the optimal values for the correlation thresholding, the two-sample t-test and finally the NSC variable selection procedures. The results of an empirical study investigating the optimal correlation threshold, two-sample t-test p-value and NSC Δ shrinkage parameter values for the base classifiers using LOOCV were reported and discussed. In this chapter, no clear selection pattern emerged for all the data sets, and no firm recommendation could be made on the strength of the various selection criteria and the accuracy of the base classifiers. The results reported in Section 5.6 indicate that the optimal threshold values for the three VS procedures for $K \in \{1, 3, 5\}$ in the KNN classifier and $C \in \{1, 1\,000, 10\,000\}$ in the SVM classifier are relatively subjective. Therefore, from the results it is evident that it is essential to perform an initial empirical study on every high-dimensional data set to determine optimal thresholding values. Furthermore, the optimal thresholding values should be determined separately for each different classifier applied to a data set.

The next chapter reports the results of the empirical study and implements the optimal tuning parameters selected for the classification procedures discussed in Section 5.6.

CHAPTER 6
RESULTS OF THE EMPIRICAL STUDY

6.1 INTRODUCTION

This chapter is devoted to a discussion and interpretation of the results of the empirical study. In this chapter, the different classification procedures summarised in Section 5.5 are applied to the six data sets and their performances are evaluated.

Data preparations for implementing the different classification procedures were done in the following way. Firstly, the data sets were randomly split into a training data set (80%) and a test data set (20%). In each case the whole process was repeated 200 times (therefore, 200 different random splits into training and test cases were performed), and the results were averaged. During this randomization, the proportions of the population groups were kept the same for all the data sets, to ensure that the relationship between the different class sizes in the resulting training and test data sets were the same as in the original data set. It is essential to keep the group proportions constant in the 200 random splits of the data sets to ensure accurate computations and low sampling bias.

The base classifiers (*viz.* KNN, SVMs and LDA) were applied to the microarray data sets and their performances in terms of classification accuracy are compared in this chapter by computing the average test error rates and the corresponding standard deviations over the 200 random splits of the data sets. For all six data sets, $K \in \{1, 3, 5\}$ was considered in the KNN classifier and the cost parameter values considered in the SVM classifier were $C \in \{1, 1\,000, 10\,000\}$. The standard errors reported in this section reflect the variability, or stability, of the classification procedures over the 200 random splits.

Note that for each data set various VS thresholding values (*i.e.* the correlation coefficient, p-value in the two-sample t-test and NSC Δ shrinkage parameter) were considered depending on the optimal value determined in Section 5.6.

The focus of this chapter is to determine the best performing classification procedure for each data set and suggest which procedure should be used in future real-world applications. However, what is meant by the best procedure is highly dependent on the context of the problem at hand. In some scenarios, minimisation of the test error rate is what constitutes an appropriate metric for measuring which classification procedure is optimal. In other scenarios, a mixture of accuracy (in terms of both test error rate and false negative rate) and interpretability is viewed as a superior way of determining the optimal procedure. For the high-dimensional microarray data sets considered in the thesis, the optimal classification procedures will be determined using the latter method.

The false negative rate (FNR), also referred to as the type II error, is used as a comparative measure in the colon cancer data set. The FNR is the probability that the biopsied colon tissue cells of a patient are classified as normal when they are in fact tumorous, *i.e.* $\Pr(\hat{Y} = 0 | Y = 1)$. All classification procedures should aim to minimise, as far as possible, the FNR, since it is more serious to classify tumorous colon tissue cells as normal than classifying normal colon tissue cells as tumorous.

It should be noted that Table 6.1 through 6.23 report the number of features used and not the number of predictors (genes) used. This is done since some dimension reduction procedures (such as fastKNN, PCA and SPCA) generate new features that are linear combinations of the original predictor variables. Therefore, although the number of features used is less than p, all p of the predictors are effectively still included in the reduced data set. Finally, the number of features used represents the modal number of features used by a specific procedure over the 200 random splits of the data set.

6.2 SUMMARY OF RESULTS

Before analysing the results obtained from the analyses described in the previous section, it is important to discuss what is expected to appear in the results.

The curse of dimensionality leads to deterioration in the performance of the KNN and other local classifiers when the number of features used is large (James *et al.,* 2013). Therefore, it is expected that the performance of the KNN classifier greatly benefits from an initial removal of irrelevant genes from the high-dimensional data sets.

As mentioned previously in Chapter 2, using a small value of K in the KNN classifier, for example $K = 1$, results in an overly flexible non-linear decision boundary, and corresponds to a very high variance but low bias classifier (James *et al.,* 2013:40). This high variance for the 1-NN classifier is due to the fact that the classification in a given region is entirely dependent on just one data case. As the value of K increases, the KNN decision boundary becomes less flexible and smoother, which corresponds to a low variance but high bias classifier as the classification of a data case is now dependent on the average of several nearest data cases. Hence, changing one data case now has a smaller effect on the classification. Therefore, in the results it is expected that the 1-NN classification procedures will have the maximum standard error (most variability between the 200 random splits), followed by the 3-NN classification procedure. The 5-NN classification procedures are expected to have the minimum standard errors. Furthermore, since KNN is a local classifier it is expected to benefit from variable selection and dimension reduction.

A large cost parameter value in the SVM classifier ($C = 1\ 000$ and $10\ 000$) results in a narrower, more wiggly and non-linear margin since the margin follows the data points more closely. Intuitively the expectation is therefore that there will be a higher standard error (and variability) reported for large values of C since the data points follow the margin more closely. A similar interpretation is expected for small values of $C = 1$ which leads to a wider and smoother decision boundary.

The RBF SVM classifier is affected by noise variables, and therefore is likely to perform better on a reduced subset of the variables in which the noise variables have been removed. However, there are scenarios where this is contradicted and the RBF SVM classifier is superior when applied to all the variables in a data set. This may be as a result of the feature selection techniques selecting the incorrect subset of variables.

6.2.1 Results on the colon cancer data set

The following tables summarise the results of the classification procedures on the colon cancer data set. The optimal threshold values implemented are those reported previously in Section 5.6. The NSC Δ shrinkage parameter value used in Table 6.1 through 6.6 is $\Delta_{opt} = 1.8932$, which was determined using the NSC classifier reported in Table 5.13.

Tables 6.1 through 6.3 report the results of the KNN classification methods for $K \in \{1, 3, 5\}$, employing a correlation threshold value of 0.2013 and a p-value $= 0.001$ in the two-sample t-test VS procedures as determined in Table 5.3 and 5.8 respectively.

The best application of the 1-NN classifier on the colon data set, occurs when the 1-NN classifier is applied to all 2 000 genes as it yields the minimum average test error rate and FNR of the nine procedures reported in Table 6.1.

Table 6.1: Results of the 1-NN procedures on the colon data set

Method	**Number of Features used**	**Test Error Rate**	**Standard Error**	**False Negative Rate**
KNN	2 000	0.2085	0.0069	0.0638
fastKNN	2	0.2927	0.0081	0.1408
ttest-KNN	34	0.2381	0.0075	0.1000
cor-KNN	706	0.2323	0.0070	0.0950
cor-fastKNN	2	0.3492	0.0086	0.1669
PCA-KNN	17	0.2188	0.0073	0.0785
ttest-SPCA-KNN	10	0.2473	0.0075	0.1096
cor-SPCA-KNN	16	0.2219	0.0067	0.0923
NSC-KNN	23	0.2523	0.0076	0.0988
Average		0.2512	0.0075	0.1051

In Table 6.1, the 1-NN classifier applied to the reduced data set consisting of ten features obtained using SPCA with the two-sample t-test yields the minimum standard error value and is therefore the most stable procedure. However, the 1-NN classifier applied to the full colon data set achieves a standard error that is only 3.73% larger.

It is clear from Table 6.1 that the 1-NN classifier is superior when applied to a larger subset features than when the data set is drastically reduced to two features as in the fastKNN and cor-fastKNN procedures.

Applying the 1-NN classifier to the subset of 17 principal components (PCA-KNN) is the second most accurate 1-NN classifier method in Table 6.1. Dimension reduction using PCA reduces the $p =$ 2 000 original genes in the colon cancer data set by 99.15%. The 99.15% reduction in the number of features used in the PCA-KNN method in Table 6.1 yields a 4.98% higher average test error rate than the best 1-NN classifier (KNN). However, from the results summarised in Table 6.1, is appears that when only one nearest-neighbour is used in the KNN classifier the classification procedures perform best using all the possible predictor values in the colon data set.

Table 6.2 below summarises the results of the 3-NN classification procedures applied to the colon cancer data set.

Table 6.2: Results of the 3-NN procedures on the colon data set

Method	**Number of Features used**	**Test Error Rate**	**Standard Error**	**False Negative Rate**
KNN	2 000	0.1800	0.0065	0.0431
fastKNN	6	0.2804	0.0080	0.1127
ttest-KNN	34	0.1900	0.0064	0.0785
cor-KNN	706	0.1646	0.0069	0.0677
cor-fastKNN	6	0.2577	0.0080	0.1227
PCA-KNN	17	0.2015	0.0072	0.0704
ttest-SPCA-KNN	10	0.1935	0.0066	0.0765
cor-SPCA-KNN	16	0.1669	0.0070	0.0727
NSC-KNN	23	0.1904	0.0063	0.0723
Average		0.2028	0.0070	0.0796

In this case the cor-KNN procedure has the minimum test error rate (0.1646) of all the 3-NN procedures reported in Table 6.2. Here cor-KNN applied the 3-NN classifier to a subset of 706 genes determined by correlation thresholding. Applying the 3-NN classifier to all 2 000 genes in the colon data set yields the minimum FNR of 0.0431, whilst the ttest-KNN procedure has the minimum standard error in Table 6.2. In Table 6.2, cor-SPCA-KNN is the second most accurate 3-NN classifier with an average test error rate that is 1.40% larger than cor-KNN. Additionally, the 3-NN classifier is applied to 97.73% less features in the reduced data using SCPA with correlation thresholding (cor-SPCA-KNN) than just correlation thresholding (cor-KNN). Remember that although the cor-SPCA-KNN procedure only makes use of 16 features, these features comprise of linear combinations of the same 706 genes that are employed in the cor-KNN procedure.

Finally, it appears that all the test error rates and FNRs decrease from Table 6.1 to 6.2 as the value of K increases from 1 to 3. Therefore, for the colon cancer data set the 3-NN classifier is superior to the 1-NN classifier.

Table 6.3 below summarises the results of the 5-NN classification methods applied to the colon data set.

Table 6.3: Results of the 5-NN procedures on the colon data set

Method	**Number of Features used**	**Test Error Rate**	**Standard Error**	**False Negative Rate**
KNN	2 000	0.2008	0.0066	0.0346
fastKNN	10	0.2400	0.0079	0.0962
ttest-KNN	34	0.1858	0.0064	0.0685
cor-KNN	706	0.1669	0.0067	0.0662
cor-fastKNN	10	0.2288	0.0069	0.1042
PCA-KNN	17	0.1988	0.0073	0.0496
ttest-SPCA-KNN	10	0.1869	0.0063	0.0731
cor-SPCA-KNN	16	0.1646	0.0069	0.0742
NSC-KNN	23	0.1796	0.0064	0.0669
Average		0.1947	0.0068	0.0704

The classification procedure with the minimum test error rate in Table 6.3 is cor-SPCA-KNN which makes use of only 16 supervised principal components (features), each of which is a linear combination of the 706 genes from correlation thresholding. However, cor-SCPA-KNN has the second largest FNR of the nine methods in Table 6.3. The results summarised in Table 6.3 indicate that the 5-NN classifier fitted to the full colon cancer data set has the minimum FNR of 0.0346. The ttest-SPCA-KNN procedure has the minimum standard error of the nine procedures reported in Table 6.3 at a value of 0.0063. The standard error of the cor-SPCA-KNN method is 9.76% larger than the minimum standard error in Table 6.3.

The results reported in Table 6.1 through 6.3 indicate that the average test error rate of the nine KNN classification procedures decreases as the value of K increases from 1 to 3 and finally to 5. This suggests that for the colon data set KNN is more accurate when more nearest neighbours are used in the classifier. Finally, from these tables it can be concluded that the most accurate KNN classification procedure on the colon data set is cor-SPCA-KNN with $K = 5$, yielding the minimum average test error rate of 0.1646. This procedure applied the 5-NN classifier to 16 supervised principal components (features), based on 706 out of the total $p = 2\ 000$ genes. Although cor-KNN with $K = 3$ also achieves the minimum test error rate of 0.1646, cor-SPCA-KNN with $K = 5$ is considered superior as it is applied to 97.73% less features and both procedures yield the same standard error of 0.0069. The FNR of the best cor-SPCA-KNN is only 2.12% higher than that of cor-KNN with $K = 3$.

The figure below graphically represents the results for the KNN classification procedures on the colon cancer data set given in Table 6.1 through 6.3.

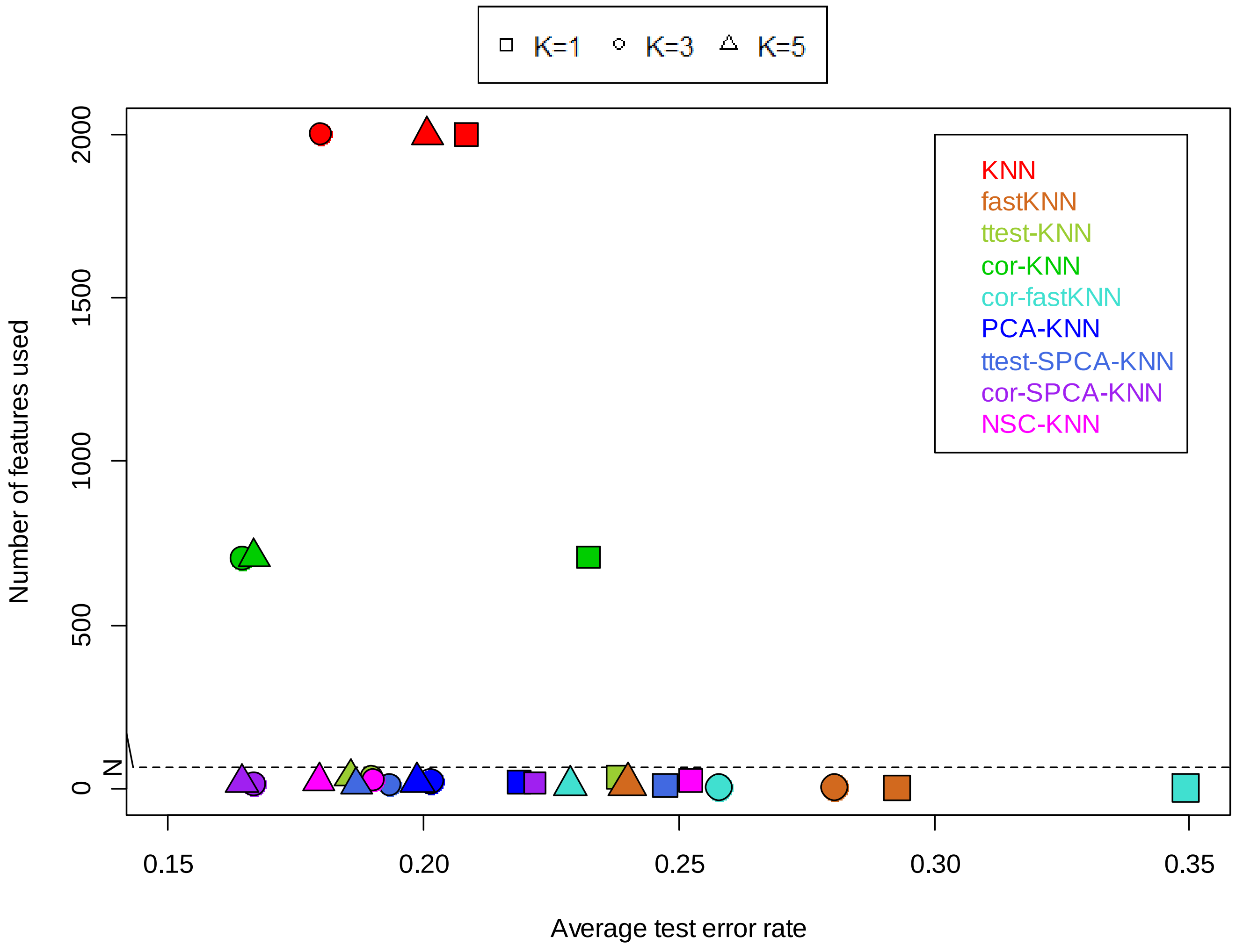

Figure 6.1: Comparison of the KNN procedures for $K \in \{1, 3, 5\}$ on the colon data

From Figure 6.1 it is clear that the most accurate KNN classifier applied to all 2 000 genes in the colon data set is obtained when $K = 3$. Figure 6.1 illustrates that for every KNN classification procedure the 1-NN classifier has a higher average test error rate (is less accurate) than the 3-NN and 5-NN classifiers. Furthermore, it is evident that cor-KNN and cor-SPCA-KNN are substantially more accurate using $K = 3$ and 5 than when $K = 1$ is employed. Additionally, Figure 6.1 shows that the 5-NN classifier is more accurate than the 3-NN classifier except when applied to all 2 000 genes and the subset of 706 genes from correlation thresholding.

Figure 6.1 also shows that the 1-NN classifier applied to the two features constructed using fastKNN feature engineering combined with correlation thresholding at a value of 0.2013, is the worst performing KNN classification method applied to the colon data set.

For the colon cancer data set, $N = 72$, which is approximately 3.60% of the number of gene expressions in the data set, $p = 2\ 000$. It is therefore difficult to determine from the above figure the exact number of features used when this number is less than p. However, Figure 6.1 illustrates that for $K \in \{1, 3, 5\}$ in the KNN classifier, only the KNN procedure applied to the full data set (KNN) and the reduced correlation thresholded data set (cor-KNN) do not overcome the curse of dimensionality as the number of features used is greater than N. The remaining KNN procedures applied to the colon data set do manage to overcome the curse of dimensionality.

Figure 6.1 confirms the result mentioned before, namely that of all the KNN procedures applied to the colon data set, cor-KNN with $K = 3$ and cor-SPCA-KNN with $K = 5$ achieve the minimum test error rate. However, cor-SPCA-KNN is preferred to cor-KNN as it is applied to 97.73% less features.

It is not clear in Figure 6.1, however, that the average standard errors of the nine KNN classification procedures displayed in Table 6.1 through 6.3 decrease as the value of K increases. These results support the expectation that the 5-NN classification procedures are the most stable over the 200 random splits whilst the 1-NN classification procedures experience the maximum variation in test error rate over the random splits yielding the maximum standard error range of 0.0067 to 0.0086 on the colon data set.

Tables 6.4 through 6.6 summarise the results of the RBF SVM classifier for $C \in \{1, 1\ 000, 10\ 000\}$ employed on the colon data set. The NSC parameter value is set at $\Delta_{opt} = 1.8932$, which was determined in Table 5.13. The optimal correlation threshold value used in Table 6.4 is 0.3026, while in Tables 6.5 and 6.6 it is set to 0.2013, which was determined in Table 5.3. Finally, the p-value for the two-sample t-test VS procedure implemented for $C = 1$ is 0.001, while for $C = 1\ 000$ and 10 000 the p-value is set to 0.01.

Table 6.4: Results of the SVM classifier with $C = 1$ on the colon data set

Method	**Number of Features used**	**Test Error Rate**	**Standard Error**	**False Negative Rate**
SVM	2 000	0.2488	0.0053	0.0292
fastKNN-SVM	10	0.2604	0.0079	0.0677
ttest-SVM	34	0.1612	0.0068	0.0715
cor-SVM	231	0.1715	0.0069	0.0754
PCA-SVM	17	0.1815	0.0068	0.0573
ttest-SPCA-SVM	10	0.1869	0.0067	0.0658
cor-SPCA-SVM	15	0.1677	0.0068	0.0604
NSC-SVM	23	0.1765	0.0067	0.0681
Average		0.1943	0.0067	0.0619

The ttest-SVM procedure is the most accurate classifier in Table 6.4, yielding an average test error rate of 0.1612 over the 200 random repetitions. The procedure uses t-*test* thresholding to reduce the colon data set to 34 genes and then fits the RBF SVM classifier with $C = 1$. The minimum FNR in Table 6.4 is 0.0292, which is achieved by applying the RBF SVM to the full data set with $p = 2\ 000$ genes.

In Table 6.4, cor-SPCA-SVM has the second smallest average test error rate, which is 4.06% larger than that of the optimal ttest-SVM procedure. However, cor-SPCA-SVM uses just less than half the number of features than ttest-SVM.

The standard errors in Table 6.4 vary from 0.0053 to 0.0079, generally being in the vicinity of 0.0068, which is generally smaller than the standard errors of the KNN classifiers in Table 6.1 through 6.3.

Table 6.5 below provides a summary of the results for the RBF SVM classification procedures with $C = 1\ 000$ on the colon data set. The standard errors reported in Table 6.5 vary from 0.0061 to 0.0080. Comparing the results in Table 6.5 to those in Table 6.4 shows that, as expected, the SVM classifier has more variation between the 200 random splits using $C = 1\ 000$ as opposed to $C = 1$. This claim is based on the fact that the average standard error over the eight SVM procedures is 3.13% higher in Table 6.5 than in Table 6.4. Similarly, the average FNR over the eight SVM procedures increased by 53.80% as the value of the C parameter increased from 1 to 1 000.

Table 6.5: Results of the SVM classifier with $C = 1\ 000$ on the colon data set

Method	Number of Features used	Test Error Rate	Standard Error	False Negative Rate
SVM	2 000	0.1696	0.0066	0.0685
fastKNN-SVM	10	0.2681	0.0080	0.1135
ttest-SVM	160	0.2054	0.0061	0.0877
cor-SVM	706	0.1708	0.0066	0.0746
PCA-SVM	17	0.2038	0.0067	0.0919
ttest-SPCA-SVM	14	0.2577	0.0075	0.1065
cor-SPCA-SVM	16	0.2219	0.0067	0.0996
NSC-SVM	23	0.2538	0.0074	0.1196
Average		0.2189	0.0070	0.0952

In Table 6.5, the SVM classifier fit to all 2 000 genes yields the minimum test error rate of 0.1696, followed by cor-SVM yielding a test error rate of 0.1708. Cor-SVM fits the SVM classifier to the reduced set of 708 genes obtained using correlation thresholding. The average test error rate over the nine SVM classification procedures increases from Table 6.4 using $C = 1$ to Table 6.5 using $C = 1\ 000$. However, the classification accuracy of the SVM classifier applied to all 2 000 genes in the colon data set improves by 31.84% as the value of the C parameter is increased from 1 to 1 000.

The results of the classification procedures using the SVM classifier with $C = 10\ 000$ on the colon data set are presented in Table 6.6.

Table 6.6: Results of the SVM classifier with $C = 10\ 000$ on the colon data set

Method	Number of Features used	Test Error Rate	Standard Error	False Negative Rate
SVM	2 000	0.1708	0.0069	0.0700
fastKNN-SVM	10	0.2746	0.0083	0.1208
ttest-SVM	160	0.2027	0.0061	0.0873
cor-SVM	706	0.1723	0.0068	0.0762
PCA-SVM	17	0.2031	0.0067	0.0919
ttest-SPCA-SVM	14	0.2573	0.0074	0.1062
cor-SPCA-SVM	16	0.2204	0.0068	0.0985
NSC-SVM	23	0.2508	0.0075	0.1177
Average		0.2190	0.0071	0.0961

In Table 6.6 there appears to be the same trend using the RBF SVM classifier with $C = 10\ 000$ as in Table 6.5 with $C = 1\ 000$. The average test error rate of the eight SVM classifiers increase only slightly by 0.04% from Table 6.5 to Table 6.6.

The standard errors reported in Table 6.6 range from 0.0061 to 0.0083, which is not substantially different from the standard errors for $C = 1\ 000$ displayed in Table 6.6. It is interesting that in the colon data set there is a greater difference in the results between using $C = 1$ and 1 000 than the difference between using $C = 1\ 000$ and 10 000 in the SVM classifier.

In Table 6.6 the most accurate method is the SVM classifier fit to all $p = 2\ 000$ genes, yielding the minimum test error rate of 0.1708 and FNR of 0.07. This is followed by cor-SVM which yields an average test error rate of 0.1723 and a FNR of 0.0762.

From the results summarised in Tables 6.4 to 6.6 it can be concluded that on average the SVM classifiers are most accurate using $C = 1$. This implies fitting a smoother boundary to the colon data set that does not regularise the variables in the model heavily. Overall the RBF SVM classifier with $C = 1$ applied to the reduced set of 34 genes from two-sample t-test thresholding is best, yielding an average test error rate of 0.1612 over the 200 random splits of the colon data set.

The results summarised in Table 6.4 through 6.6 are plotted in the figure below. The three different cost parameter values used in the RBF SVM classifier are represented using three different symbols in this figure.

In Figure 6.2 it is clear that when applied to all 2 000 genes in the colon data set, the RBF SVM classifier is more accurate using $C = 1\ 000$ or 10 000 than with $C = 1$. However, Figure 6.2 illustrates that when applied to a reduced subset of the colon data set, the RBF SVM classifier with $C = 1$ is superior to a RBF SVM with $C = 1\ 000$ or 10 000. Therefore, from the results displayed in Figure 6.2 it can be concluded that on average the SVM classifier with $C = 1$ is the most accurate in terms of test error rate.

From Figure 6.2 it appears that there are two optimal C parameter values. A larger C parameter value, either 1 000 or 10 000, performs better on all 2 000 genes in the colon data set. On the other hand using a smaller C parameter value ($C = 1$) is better on a reduced set of gene expressions.

Figure 6.2 supports the earlier conclusion that overall the RBF SVM classifier with $C = 1$ applied to the subset of 34 genes obtained from the two-sample t-test VS procedure is the best SVM option for the colon cancer data set.

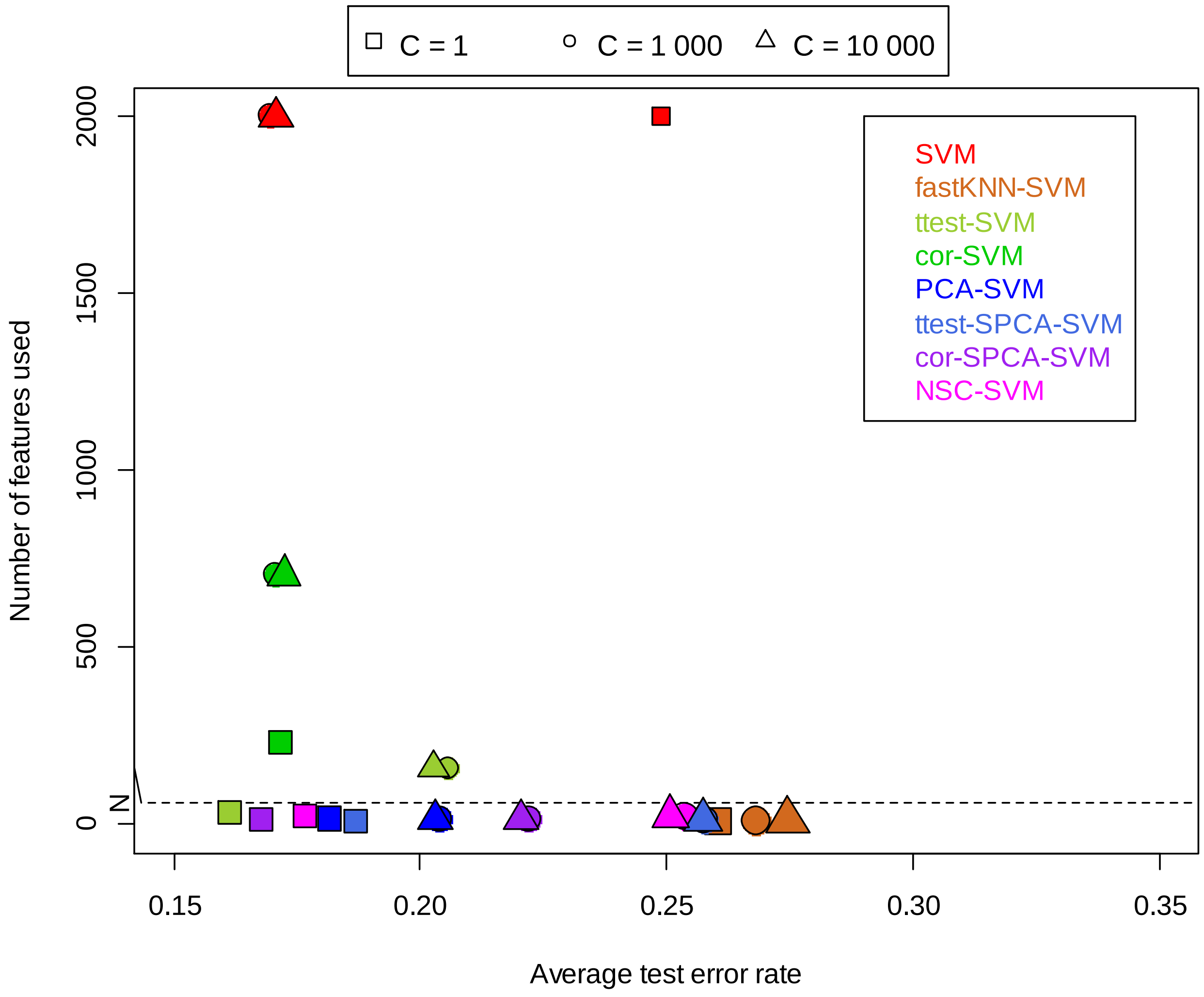

Figure 6.2: Comparison of the SVM classification methods on the colon data set

Table 6.7 provides the results of the LDA based classification methods on the colon data.

Table 6.7: Results of the LDA based classification methods on the colon data

Method	Number of Features used	Test Error Rate	Standard Error	False Negative Rate
NSC-LDA	23	0.2581	0.0085	0.1200
NSC	23	0.1950	0.0061	0.0592
DLDA	2 000	0.3458	0.0120	0.2612
NSC-DLDA	23	0.1762	0.0074	0.0808
Average		0.2437	0.0085	0.1303

In Table 6.7 the NSC VS procedure results in a 98.85% reduction in the number of genes used (only 23 genes). NSC-DLDA is the best classification procedure in Table 6.7, achieving an average test error rate of 0.1762 over the 200 repetitions.

From Table 6.7 it is clear that the performance of the DLDA classifier improves once the irrelevant genes are removed. The DLDA procedure applied to the reduced subset of 23 genes yields a test error rate that is 49.05% lower than that of DLDA applied to the full colon cancer data set. It is interesting that the DLDA classifier is 9.66% more accurate than the NSC classifier applied to the same subset of 23 genes obtained using the NSC VS procedure.

Looking at the results summarised in Tables 6.1 through 6.7, there are several general conclusions that can be made. It is interesting to note that the minimum FNR achieved by the KNN classification procedures on the colon data set is 0.0346, which is yielded by the 5-NN classifier applied to all the genes. The RBF SVM classifier with $C = 1$ applied to all the genes in the colon data set achieves the overall minimum FNR rate of 0.0292 of all the classification procedures displayed in Table 6.1 through 6.7. The SVM classifier with $C = 1$ also yields the minimum standard error, of 0.0053 suggesting that it has the least variation in test error rate between the 200 random splits.

The results summarised in Table 6.1 through 6.7 suggest that the most accurate classification procedure on the colon cancer data set is the ttest-SVM with $C = 1$, yielding an average test error rate of 0.1612 over the 200 random splits. However, this procedure yields a FNR that is 144.74% larger than the minimum FNR achieved by the SVM classifier applied to all the genes in the colon data set.

The worst classification procedure applied to the colon cancer data set is cor-fastKNN with $K = 1$, yielding an average test error rate of 0.3492 over the 200 random splits.

The following figure plots the results for the 5-NN classification procedures (given in Table 6.3) and the RBF SVM classification procedures with $C = 1$ (given in Table 6.4). It should be noted that the test error rate of classifying all the test observations in the full colon data set to the majority class is approximately 0.3548.

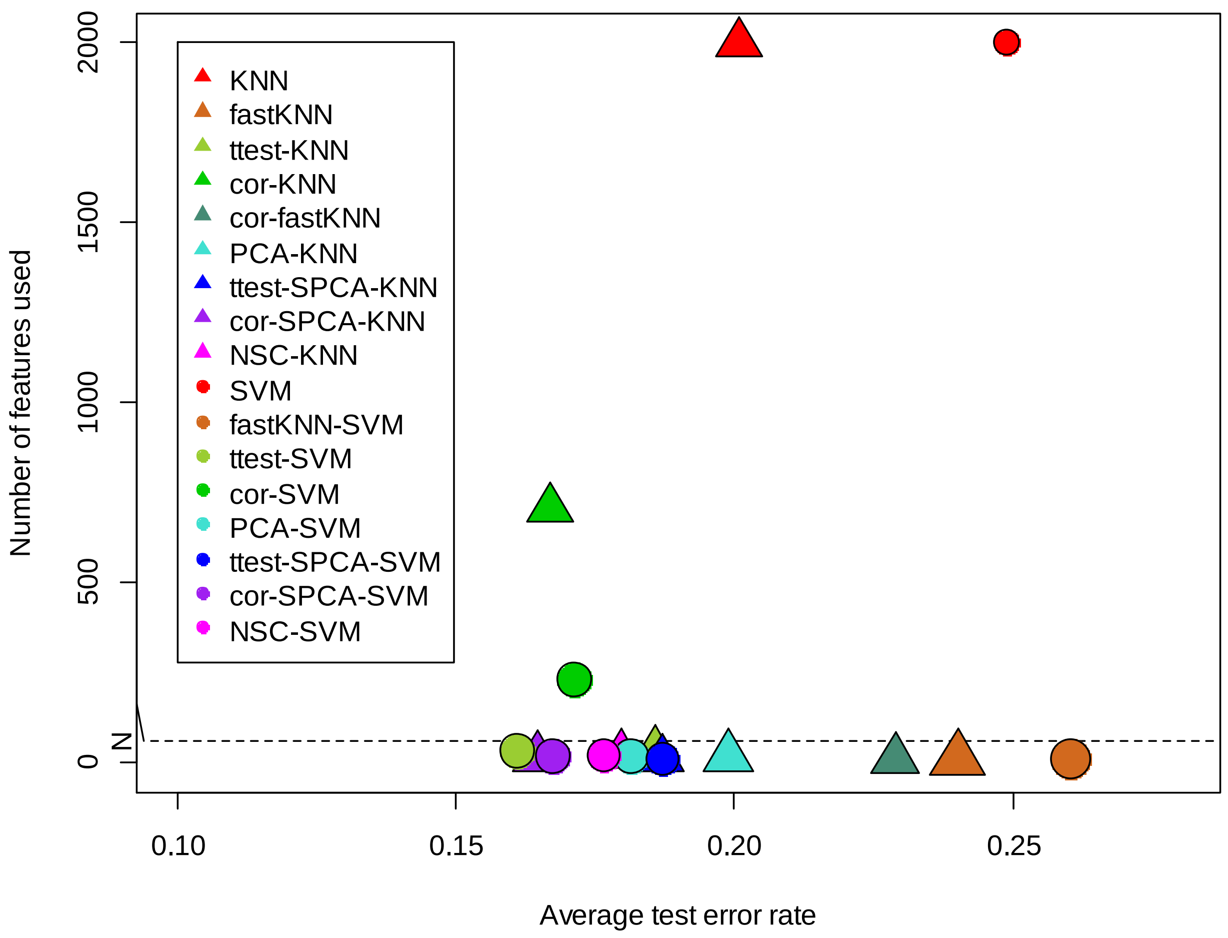

Figure 6.3: Comparison of the 5-NN and SVM (with $C = 1$) procedures on the colon data set

From Figure 6.3 it is clear that when using all 2 000 genes in the colon data set, the 5-NN classifier is more accurate than the RBF SVM classifier with $C = 1$.

Figure 6.3 illustrates that applying the 5-NN and SVM classifiers to the subset of ten features obtained using fastKNN with $K = 5$ is the worst performing feature extraction technique. However, as expected, the 5-NN classifier is more accurate than the SVM classifier when applied to the subset of ten features obtained using fastKNN with $K = 5$ as a feature extraction method.

The sizes of the points in Figure 6.3 represent the standard errors of the different classification methods. Due to the small standard errors the point sizes are magnified by 400. From Figure 6.3 it is evident that using fastKNN as a feature engineering procedure results in the largest variation of the test error rate between the 200 random splits for both the KNN and SVM classifier. On the other hand SVM applied to all the genes in the colon data set has the smallest point and is the most stable procedure plotted in Figure 6.3.

Finally, from Figure 6.3 it is evident that the two-sample t-test yields the most accurate subset of genes for the SVM classifier with $C = 1$. However, using SPCA with correlation thresholding (cor-SPCA-KNN) leads to the most informative subset of genes for the 5-NN classifier.

The goal of analysing the colon cancer data set is to classify and predict the diagnostic category of a colon tissue sample based on its gene expression profile. However, not only does a procedure need to accurately diagnose whether adenocarcinoma is present in a tissue sample, it also has to determine which gene expressions are important and useful in distinguishing between tumorous and normal samples. The following figures therefore attempt to identify the important/ significant predictor variables in the colon data set. An important predictor variable is defined as a gene that is selected frequently (in this case selected in at least 80% of the 200 random repetitions) using the three different feature extraction methods (*viz.* correlation thresholding, the two-sample t-test and the NSC procedure). The genes selected frequently by these VS procedures are plotted in Figure 6.4 through 6.6 below. The selection percentage threshold is kept constant at 80% of the random splits and therefore the probability that a gene will be selected given a random split will be above 80%.

Figure 6.4 gives the index and selection percentage of the genes selected frequently over the 200 random repetitions based on implementing the two optimal p-values (0.001 and 0.01) for two-sample t-test thresholding on the colon cancer data set. It is evident that using a larger (more liberal) p-value results in a higher number of genes being selected. This is what is intuitively expected.

The left panel in Figure 6.4 displays the 14 gene indices from the colon cancer data set that are considered significant at a p-value$=$ 0.001. These 14 genes are therefore considered useful in predicting the diagnostic category of a sample based on its gene expression profile. Of the 14 gene indices displayed, 8 genes appear 100% of the time using two-sample t-test thresholding.

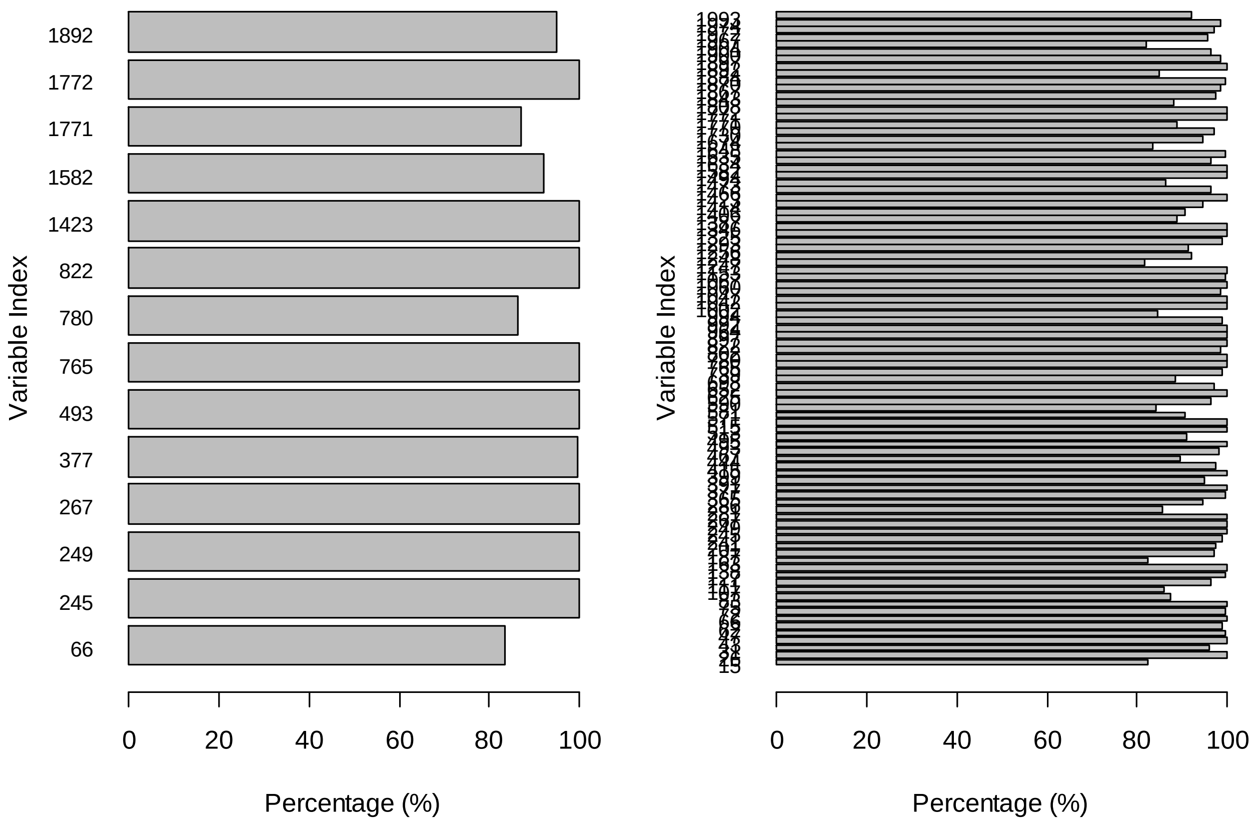

Figure 6.4: Variables selected using the two-sample t-test with a p-value of 0.001 (left) and 0.01 (right) on the colon data set

The right panel in Figure 6.4 shows that performing VS on the colon data based on the two-sample t-test approach with p-value = 0.01 leads to much fuller models being selected. Now there are 90 genes in the colon data set that are selected in at least 80% of the random splits. Of the 90 genes plotted in the right panel of Figure 6.4, 31 genes have a 100% selection frequency.

Additional investigations revealed that when implementing the two-sample t-test at a p-value = 0.001 on the colon data set (as in the left panel of Figure 6.4), 1 738 (86.90%) of the $p =$ 2 000 genes are never selected and are thus considered irrelevant in distinguishing between normal and tumorous colon tissue samples.

All 14 significant genes plotted in the left-hand panel of Figure 6.4 also appear in the list of 90 significant genes in the right panel of Figure 6.4. Furthermore, the 14 genes that are plotted in the left-hand panel all have a 100% selection frequency in the right-hand panel and make up 45.16% of the 31 genes that are selected 100% of the time with a p-value = 0.01.

The papers by Alladi *et al.* (2008) and Kulkarni *et al*. (2011) list the top ten genes from the colon data set selected by the t-statistic. Analysis revealed that only five genes in this list appear in the left panel of Figure 6.4 and of these five genes only the genes with indices 1 772 and 249 are selected 100% of the time. However, all ten genes are selected in every random split in the right panel of Figure 6.4. The papers by Alladi *et al.* (2008) and Kulkarni *et al*. (2011) both assume unequal variances in the computation of the t-statistic, while in the thesis the t-statistic is computed using the pooled within-group standard error.

It should be noted that the 14 genes displayed in the left panel of Figure 6.4 are the subset of genes selected at least 80% of the time from the subset of 34 genes selected by the two-sample t-test with a p-value $=$ 0.001. The RBF SVM classifier with $C = 1$ applied to the subset of 34 genes is the best classification procedure on the colon data set.

The figure below displays the indices of the genes in the colon data set frequently selected by correlation thresholding at the optimal values 0.2013 and 0.3026.

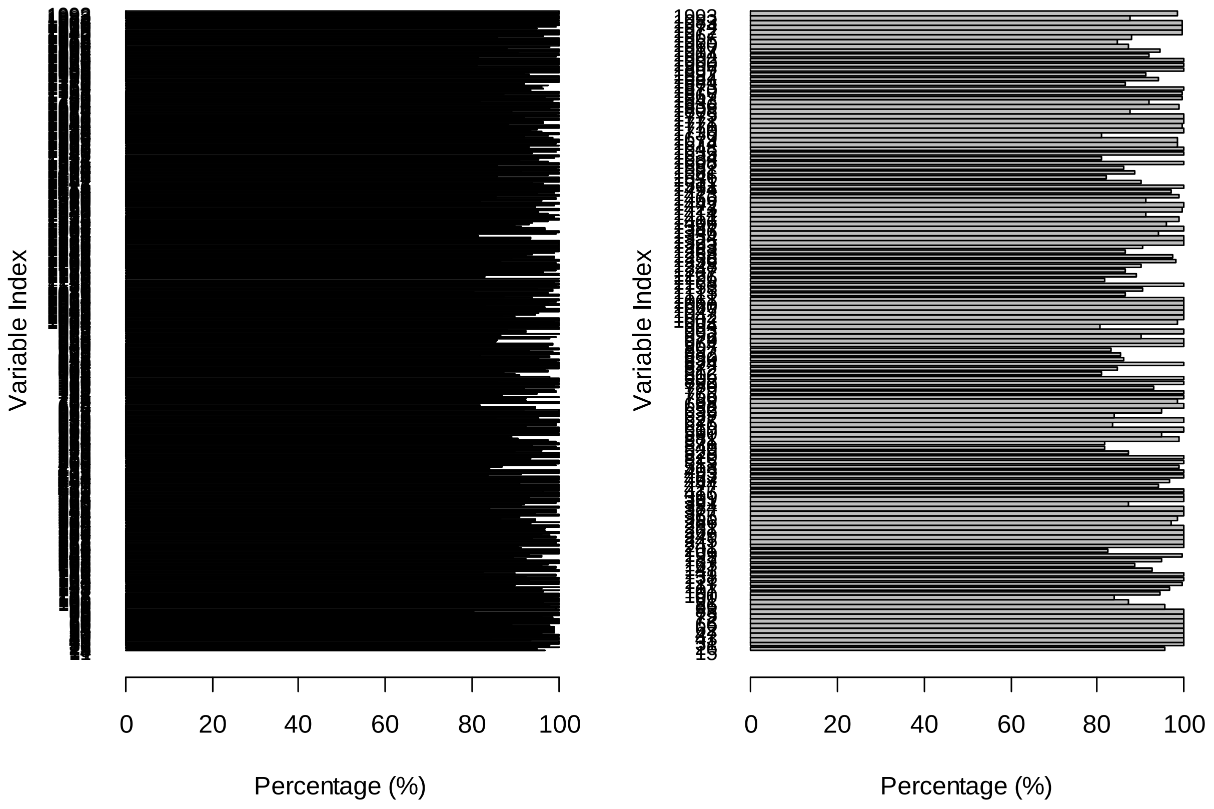

Figure 6.5: Variables selected using correlation thresholding at a value of 0.2013 (left) and 0.3026 (right) on the colon data set

Figure 6.5 illustrates that applying correlation thresholding at a smaller value to the colon data set leads to much fuller models being selected. Consequently, Figure 6.5 is very difficult to interpret since 394 genes are selected at least 80% of the time when performing correlation thresholding at 0.2013 (left panel) and 137 genes are selected in at least 80% of the splits when implementing correlation thresholding at a value of 0.3026 (right panel). In the left panel of Figure 6.5, 178 of the 394 genes displayed (45.18%) are selected 100% of the time by correlation thresholding whereas in the right panel 56 genes (40.88%) are selected in every one of the 200 random splits.

Additional analysis of the results plotted in Figure 6.5 shows that out of the $p = 2\ 000$ genes in the colon data set, 138 and 985 genes are never selected in the 200 random splits when correlation thresholding is implemented at values of 0.2013 and 0.3026 respectively. These subsets of genes are therefore considered irrelevant in distinguishing between tumorous and normal colon tissue samples.

The analysis reported in Table 5.3 revealed that in the colon data set the gene with index 249 has the maximum correlation with Y. In both the left and right panel of Figure 6.5, correlation thresholding selects gene index 249 in every one of the 200 random splits.

Figure 6.6 presents the index and selection percentage of the variables selected in at least 80% of the 200 random splits of the colon data set using the NSC procedure with Δ_{opt}= 1.8932. Note that the results summarised in Table 6.7 indicate that when the NSC procedure is applied to the colon data set with $\Delta\ = 1.8932$, the modal frequency occurred at 23 genes.

Of the 11 genes displayed in Figure 6.6, five are selected in every split. Although it is not indicated in Figure 6.6, further analysis revealed that 1 908 genes in the colon data set are never selected in any of the 200 random splits using the NSC procedure, implying that they are irrelevant in distinguishing the diagnostic category of a sample based on its gene expression profile.

The five genes that are selected 100% of the time in Figure 6.6 (the genes with indices 245, 249, 765, 822 and 1423) are also selected 100% of the time in Figure 6.4 and Figure 6.5. Therefore, the results displayed in Figure 6.4 through 6.6 suggest that in the colon cancer data set the genes with indices 245, 249, 765, 822 and 1423 are highly significant in predicting the diagnostic category of a sample based on its gene expression profile.

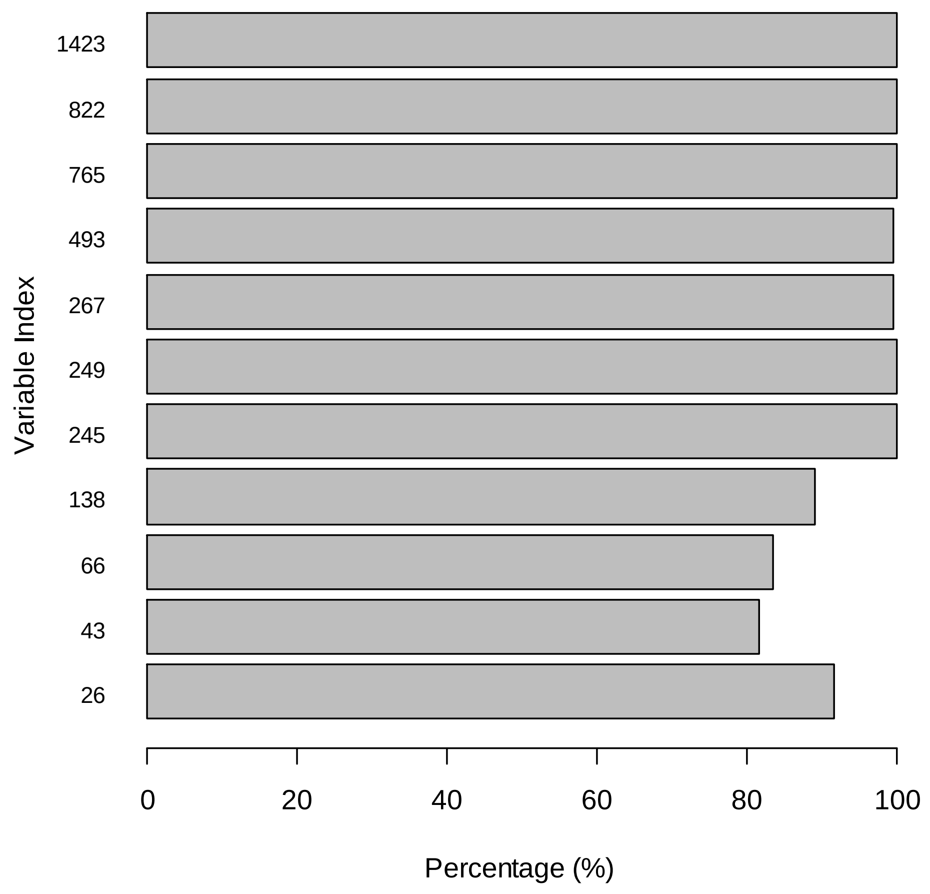

Figure 6.6: Variables selected using the NSC procedure with $\Delta = 1.8932$ on the colon data set

Additionally, in the colon data set the genes with indices 66, 267 and 493 also appear in Figure 6.4 through 6.6, which indicates that all three variable selection procedures select these three genes in at least 80% of the 200 repetitions. It can therefore be concluded that the genes with indices 66, 267 and 493 are vital in distinguishing tumorous from normal colon tissues in the colon cancer data set.

As mentioned previously the most accurate classification procedure for the colon data set is the RBF SVM classifier with $C = 1$ applied to the subset of 34 genes selected using a two-sample t-test with a p-value $= 0.001$. However, Figure 6.4 illustrated that only 14 genes from the colon cancer data are selected by two-sample t-test thresholding (with a p-value $= 0.001$) at least 80% of the time over the 200 random splits. These genes should therefore be considered important when distinguishing between the two population groups. Therefore, the indices of the 14 genes displayed in the left panel of Figure 6.4 and their GenBank accession numbers and description are given in the table below.

Table 6.8: Description of the important gene expressions in the colon data set

Gene Index	GenBank Accession Number	Gene Expression	Freq. (%)
66	T71025	H.sapiens cDNA similar to gb:J03910_rna1 human	83.50
245	M76378	Human cysteine-rich protein (CRP) gene, exons 5 and 6.	100.00
249	M63391	Human desmin gene, complete cds (DES).	100.00
267	M76378	Human cysteine-rich protein (CRP) gene, exons 5 and 6.	100.00
377	Z50753	H.sapiens mRNA for GCAP-II/uroguanylin precursor.	99.50
493	R87126	Myosin heavy chain, nonmuscle (Gallus gallus)	100.00
765	M76378	Human cysteine-rich protein (CRP) gene, exons 5 and 6.	100.00
780	H40095	Macrophage migration inhibitory factor (MIF) (Human)	86.50
822	T92451	Tropomyosin, fibroblast and epithelial muscle-type (Human)	100.00
1 423	J02854	Myosin regulatory light chain 2, smooth muscle Isoform (Human);contains element TAR1 repetitive element	100.00
1 582	X63629	H.sapiens mRNA for p cadherin.	92.00
1 771	J05032	Human aspartyl-tRNA synthetase alpha-2 subunit mRNA, complete cds.	87.00
1 772	H08393	Collagen alpha 2(XI) Chain (Homo sapiens)	100.00
1 892	U25138	Human MaxiK potassium channel beta subunit mRNA, complete cds.	95.00

The information reported in Table 6.8 and additional information regarding the remaining 1 986 genes in the colon cancer data set is available from http://genomics-pubs.princeton.edu/oncology/affydata/. Gene index 66 has the minimum selection frequency (167 out of 200 random splits) of the 14 genes displayed in Table 6.8.

As mentioned previously, five of the genes displayed in Table 6.8 (*viz.* the genes with indices 249, 780, 1 582 1 771, and 1 772) also appear in the list of ten best genes selected by t-statistic thresholding in the papers by Alladi *et al.* (2008) and Kulkarni *et al.* (2011).

Note that in Table 6.8 the genes with indices 245, 267 and 765 all have the same GenBank Accession Number, namely M76378. It is interesting that out of all the $p =$ 2 000 genes in the colon data set all three genes that have the GenBank Accession Number M76378 are selected as significant in discriminating between the tumorous and normal groups by the two-sample t-test VS procedure. Cherian *et al.* (2003) explain that the human cysteine-rich protein (CRP) which has the GenBank Accession Number

M76378 has been associated with protection against DNA damage, oxidative stress and apoptosis. Yap *et al.* (2004) reported a down-regulation of this gene in tumour tissue stating that it suggests the lack of protection against DNA damage and that therefore these gene have discriminative power. This supports the findings that genes 245, 267 and 765 were correctly identified as significant in Table 6.8.

The paper by Kulkarni *et al.* (2011:2756) refers to the study by Jiang *et al.* (2008) which suggests that the t-test VS procedure correctly identified gene index 249 as significant. The study by Jiang *et al.* (2008) concluded that gene index 249 is down regulated in colon cancer samples (which means that gene 249 has a lower expression in tumorous samples than in normal samples). Note that in Section 5.6.2, the gene with index 249 is identified as the most significant gene by both the two-sample t-test and correlation coefficient, and thus it is reassuring to see that it is selected 100% of the time by the VS procedure in Table 6.8.

The identification of gene 377 as important is supported by Notterman *et al.* (2001), who showed that a reduction of uroguanylin in gene index 377 could be an indication of colon tumours (gene 377 is up-regulated in normal colon tissue). Additionally, Shailubhai *et al*. (2000) reported that treatment with uroguanylin has been shown to have positive therapeutic significance with reduction in the number of pre-cancerous colon tumours, shrinkage in the remaining ones and observed apoptosis (death) of adenocarcinoma cells.

Yap *et al.* (2004) state that gene 780 functions as a pluripotent cytokine involved in broad-spectrum pathophysiological events in association with inflammation and immune responses. The papers by Yap *et al.* (2004) and Campa *et al.* (2003) suggest that gene 780 is involved in tumorigenesis (the production or formation of a tumour or tumours) and is therefore correctly identified as important in Table 6.8.

A study by Kishino *et al.* (2000) found genes 822 and 1 892 (GenBak Accession Numbers T92451 and U25138 respectively) to be more highly expressed in normal colon tissue samples than tumorous tissue samples. Hence, there is a biological reason to support the VS method in Table 6.8 selecting genes 822 and 1 892 in the colon data set as significant.

Table 6.8 displays gene 1 423 as significant, which is in accordance with the original analysis of the data set by Alon *et al.* (1999). Furthermore Keely *et al.* (1998) state that there is a biological interpretation for the discriminative power of gene 1 423 as it is known to be an intracellular target of integrins which affects cell motility. Subsequently, the invasiveness and metastatic potential of tumour cells is strongly dependent on this motile activity. Similarly, Yam *et al.* (2001) state that gene 493 is known as a component of cytoskeletal network and acts as a tumour suppressor.

Gene 1 772 is displayed in Table 6.8 as important in distinguishing between normal and tumorous colon tissue samples. This finding supports the conclusion made by Fischer *et al.* (2001) who state that collagen 11, a heterotrimeric molecule consisting of $\alpha 1, \alpha 2$ and α3 chains, have a role in the formation of collagen fibrils and gene index 1 772 is a gene for collagen which is normally not expressed in adult colon tissue, but was found to be expressed in colorectal carcinomas.

According to Boulesteix (2004), the 5 top-ranking genes in the colon cancer data set using PLS are genes 493, 377, 249, 1 635 and 1 423. Besides gene 1 635 which is selected in only 134 (67%) of the 200 random splits, the remaining four genes are selected in at least 199 of the 200 random splits by the t-test VS procedure.

In the papers by Alladi *et al.* (2008) and Kulkarni *et al.* (2011), several other classification procedures are also studied, which are not investigated in this thesis. The paper by Ben-Dor *et al.* (2000) achieved an accuracy of 80.60% when applying the nearest neighbour classifier to all the genes in the colon data set. However, in this thesis the most accurate KNN classifier applied to all the genes in the colon data set is with $K = 3$ yielding an accuracy of 82.00% over the 200 random splits. Therefore, the KNN classification procedure with $K =$ 3 applied to all the genes implemented in this thesis is slightly superior to the nearest neighbour classifier applied to all the genes in the paper by Ben-Dor *et al.* (2000).

It should be noted that in the paper by Alladi *et al.* (2008), the authors use a different base classifier called Genetic Programming and applied it to the genes obtained using mutual information based feature selection which predicts the validation colon data set with 100% accuracy. The reader is referred to this paper for more details in this regard.

In this thesis the best classification procedure applied to the colon cancer data set is the two-sample t-test based feature selection with RBF SVM classifier with $C =$ 1 achieving the maximum classification accuracy of 83.88% over the 200 random splits of the colon data set.

6.2.2 Results on the leukemia data set

Tables 6.9 through 6.11 summarise the results of the different classification procedures applied to the leukemia data set. In these tables the thresholding values and shrinkage parameters applied to the leukemia data set are those determined as optimal values from the LOOCV described in Section 5.6. In Table 6.9 through 6.11, the NSC shrinkage parameter value is set to $\Delta_{opt} = 2.1597$ and the p-value in the two-sample t-test is set to 0.001. The correlation coefficient for thresholding in the KNN and SVM classifiers is set to 0.1000 in Table 6.9 and 6.11, except for the 5-NN classifier in Table 6.9 where the correlation thresholding is implemented at a value of 0.2469. Data splitting was once again repeated 200 times and the average number of features used and the mean and standard error of the test error rates were calculated for each classification procedure.

Note that the false negative rate is not computed for the leukemia data set, since neither of the population groups represent normal tissue and it is not more serious to misclassify one population group compared to the other.

Table 6.9: Results of the KNN classification procedures on the leukemia data set

Method	**$K = 1$**			**$K = 3$**			**$K = 5$**		
	No. of feature	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**
KNN	3 571	0.0443	0.0034	3 571	0.0277	0.0028	3 571	0.0313	0.0031
fastKNN	2	0.0897	0.0050	6	0.0653	0.0040	10	0.0587	0.0040
ttest-KNN	538	0.0133	0.0019	538	0.0130	0.0019	538	0.0130	0.0019
cor-KNN	2 643	0.0300	0.0030	2 643	0.0290	0.0029	1 345	0.0190	0.0022
cor-fastKNN	2	0.0620	0.0042	6	0.0500	0.0036	10	0.0207	0.0023
PCA-KNN	42	0.0347	0.0032	42	0.0360	0.0032	42	0.0333	0.0031
ttest-SPCA-KNN	34	0.0133	0.0019	34	0.0130	0.0019	34	0.0130	0.0019
cor-SPCA-KNN	41	0.0227	0.0025	41	0.0323	0.0030	38	0.0183	0.0022
NSC-KNN	25	0.0340	0.0029	25	0.0347	0.0032	25	0.0347	0.0029
Average		0.0382	0.0031		0.0334	0.0029		0.0269	0.0026

The average test error rate of the KNN classification procedures over the 200 random splits of the leukemia data set ranges from 0.0130 to 0.0897, which is a relatively small interval. The results in Table 6.9 support the statement by Golub *et al.* (1999) that the leukemia data set is considered a low error rate data set. In Table 6.9, the standard errors vary from 0.0019 to 0.0050, which is small and indicates that there is not high variability in the performance of the KNN classification procedures over the 200 random splits. As expected, the 5-NN classification procedures have the minimum average standard errors of 0.0026, followed by the 3-NN classification procedures (0.00290) and the 1-NN classification procedures, which have the most variation and maximum average standard error of 0.0031.

It is evident from Table 6.9 that in terms of both test error rate and standard error the best 1-NN, 3-NN and 5-NN classification procedures are the ttest-KNN and ttest-SPCA-KNN procedures. However, the ttest-KNN and ttest-SPCA-KNN achieve a 2.50% lower average test error rate using $K = 3$ and 5 instead of 1 NN. Although for $K = 3$ and 5, the ttest-KNN and ttest-SPCA-KNN both yield an average test error rate of 0.0130, ttest-SPCA-KNN applied the KNN classifier to 34 supervised PCs (features) which is 93.68% less features than the 538 genes used in the ttest-KNN. The results in Table 6.9 show that the ttest-SPCA-KNN procedure with $K = 3$ and 5 is the best KNN procedure in which the number of features used is less than $N = 72$.

The worst performing procedure reported in Table 6.9 is fastKNN with K =1, yielding the maximum average test error rate of 0.0897 as well as the maximum standard error of 0.005 over the 200 repetitions. In Table 6.9 it is evident that the overall performance of the fastKNN procedure as both a feature extraction and classification method improves as the value of K increases from 1 to 5.

Note that in general, apart from when applied to the PCA and SPCA dimension reduction techniques, the KNN classifier's overall performance improves as K increases from 1 to 3. It is clear from the entries in the last row of Table 6.8 that the average test error and average standard error of the KNN classifier both decrease as K increases. Therefore, the KNN procedure applied to the leukemia data set is more accurate and experiences less variation between the random repetitions when 5 nearest neighbours are considered.

From Table 6.9 it can be concluded that when applied to the leukemia data set, the KNN classifier greatly benefits from an initial elimination of irrelevant genes using the two-sample t-test VS procedure. This may be explained by the local nature of the KNN classifier.

This section now investigates the performance of the SVM classifier on the leukemia data set. The results in Section 5.6 show that for $C \in \{1, 1\ 000, 10\ 000\}$ in the SVM classifier applied to the leukemia data set, the optimal correlation thresholding value is 0.1000 and the optimal p-value in the two-sample t-test thresholding procedure is 0.001. The optimal p-value in the two-sample t-test, the optimal correlation value, the optimal NSC shrinkage parameter value Δ and the value of K used in the fastKNN feature extraction method are all the same for C =1, 1 000 and 10 000. Hence, the number of features used in each SVM classification procedure is constant for $C \in$ {1, 1 000, 10 000}.

Table 6.10: Results of the SVM classification procedures on the leukemia data set

Method	**No. of features**	**$C = 1$**		**$C = 1\,000$**		**$C = 10\,000$**	
		Test Error	**Std. Error**	**Test Error**	**Std. Error**	**Test Error**	**Std. Error**
SVM	3 571	0.0110	0.0018	0.0107	0.0017	0.0107	0.0017
fastKNN-SVM	10	0.0480	0.0037	0.0777	0.0043	0.0763	0.0043
ttest-SVM	538	0.0133	0.0019	0.0177	0.0021	0.0183	0.0021
cor-SVM	2 643	0.0110	0.0018	0.0110	0.0018	0.0110	0.0018
PCA-SVM	42	0.2230	0.0047	0.0213	0.0025	0.0213	0.0025
ttest-SPCA-SVM	34	0.1440	0.0053	0.0307	0.0029	0.0307	0.0029
cor-SPCA-SVM	41	0.2090	0.0050	0.0210	0.0025	0.0210	0.0025
NSC-SVM	25	0.0340	0.0033	0.0307	0.0029	0.0303	0.0029
Average		0.0867	0.0034	0.0276	0.0026	0.0275	0.0026

From Table 6.10 it can be observed that the most accurate and stable SVM classification procedure on the leukemia data set is the RBF SVM with the C parameter equal to both 1 000 and 10 000 fitted to all 3 571 genes achieving the minimum test error rate of 0.0107. Since there is a tie between the SVM fitted to the full leukemia data set for C =1 000 and 10 000, the RBF SVM with C = 1 000 fitted to the full data set is chosen as the best SVM method in Table 6.10. For all values of C, the cor-SVM procedure yields the second lowest test error rate of 0.0110. Comparing these two procedures for C = 1 000 and 10 000, cor-SVM is applied to 2 643 genes which is a 25.99% reduction from the total number p = 3 571 of genes used in the best SVM procedure. Nevertheless, cor-SVM's average test error rate is only 3.12% larger than the average test error rate of the SVM procedures.

It is known that the RBF SVM classifier is affected by noise variables, consequently it is likely to perform better on a reduced subset of variables in which the noise variables have been removed. However, the results in Table 6.10 are counter initiative as it appears that for $C \in$ {1, 1 000, 10 000} the RBF SVM classifier is superior when applied to all 3 571 genes (SVM) rather than to any reduced subset of the genes. This may be as a result of the feature selection and dimension reduction techniques implemented in Table 6.10 selecting incorrect subsets of genes.

From the results reported above it appears that the RBF SVM classifier, especially when $C = 1$, does not perform well using subsets of genes from linear feature extraction methods such as in PCA, SPCA and fastKNN.

The standard errors of the SVM procedures in Table 6.10 vary from 0.0018 to 0.053, which is the same general variation as observed for the KNN classification procedures on the leukemia data set displayed Table 6.9. As mentioned previously it is intuitively expected that the SVM classifier will exhibit more variation (a higher standard error) for larger values of the cost parameter. However, due to the abnormally large standard errors (and test error rates) for the PCA-SVM, ttest-SPCA-SVM and cor-SPCA-SVM procedures when C =1, the average standard error over the eight SVM procedures displayed in Table 6.10 is 32.60% larger for $C = 1$ than for $C = 1\ 000$ and 10 000.

The average test error and average standard error of the eight classification procedures given in the last row of Table 6.10 both decrease as the value of the C parameter increases. Note that there is a substantial decrease of 68.17% in the average test error rate as the C parameter increases from 1 to 1 000. However, as C further increases from 1 000 to 10 000 the average test error rate only decreases by 0.45%. Therefore, using the results summarised in Table 6.10, it appears that for the leukemia data set using a larger value for the parameter C in the RBF SVM classifier is preferable.

It is interesting to note that for the SVM classifier the optimal correlation threshold value of 0.1000 was selected at least 96% of the time reported in Table 5.4. Furthermore in Table 6.10, cor-SVM achieved a consistent test error rate (small standard error of 0.0018) for all values of C. On the other hand, the KNN classifier's selection frequencies is dispersed over the correlation candidate values for the 100 random splits and there is no single thresholding value that is selection majority of the time for all values of K. Subsequently the KNN classifier yields a higher standard error for cor-KNN in Table 6.9 ranging from 0.0022 to 0.0300. For KNN, the t-*test* variable selection procedure selects the optimal p-value of 0.001 majority (72%) of the time in Table 5.9 and on average the standard errors for $K \in \{1, 3, 5\}$ are more stable and experiences 31.07% less variation than cor-KNN in Table 6.9.

From Table 6.9 and 6.10 it can be concluded that on average when applied to the leukemia data set the KNN classifier is most accurate with $K = 5$ and the RBF SVM is most accurate with $C = 10\ 000$. Therefore, Figure 6.7 below compares the results of the 5-NN and RBF SVM with $C = 10\ 000$ classification procedures on the leukemia data set.

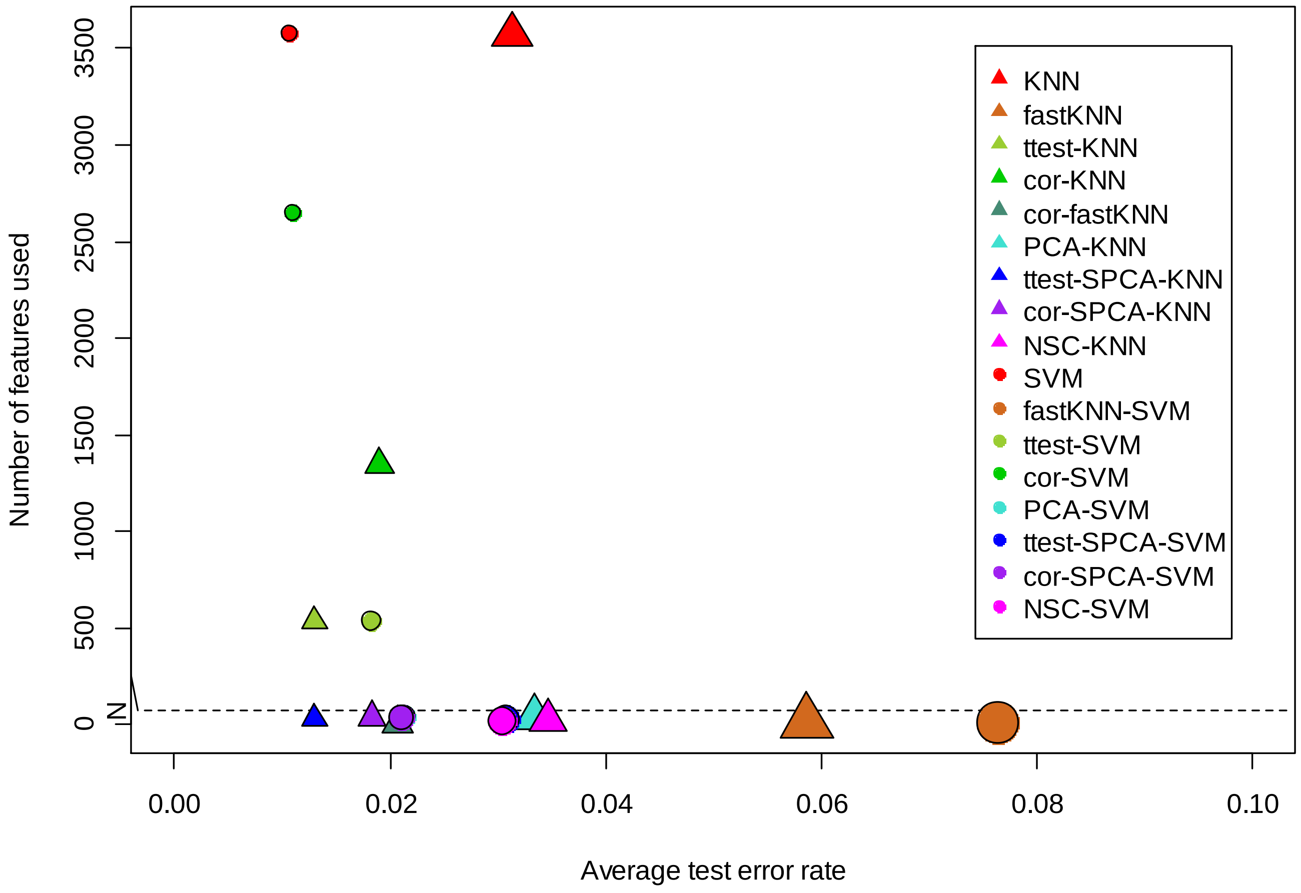

Figure 6.7: Comparison of the 5-NN and SVM (with $C = 10\ 000$) procedures on the leukemia data set

Figure 6.7 confirms that the most accurate procedure on the leukemia data set is the RBF SVM classifier applied to all the genes (SVM) with the minimum test error rate of 0.0107. This is followed by cor-SVM with an average test error rate of 0.0110. Nevertheless, the points for both SVM and cor-SVM plotted in Figure 6.7 appear above the horizontal dotted line at $N = 72$. Therefore, looking at only the procedures in Figure 6.7 that make use of less than N features, the ttest-SPCA-KNN procedure is the most accurate, yielding a classification accuracy of 98.70% over 200 random splits. Figure 6.7 illustrates that for the 5-NN classifier, ttest-KNN and ttest-SPCA-KNN both achieve the minimum test error rate of 0.0130 yet ttest-SPCA-KNN uses substantially less features than ttest-KNN.

The sizes of the points in Figure 6.7 represent the standard errors of the different classification methods. Due to the small standard errors the point sizes are magnified by 800 which is 2 times larger than the magnification used for the colon data set in Figure 6.3. From Figure 6.7 it is evident that using fastKNN as a feature engineering procedure results in the largest variation of the test error rate between the 200 random splits for both the KNN and SVM classifier.

Table 6.11 provides the results of the LDA based classification methods on the leukemia data set.

Table 6.11: Results of the LDA based classification methods on the leukemia data

Method	**No. of Features used**	**Test Error Rate**	**Standard Error**
NSC-LDA	25	0.0627	0.0043
NSC	25	0.0543	0.0034
DLDA	3 571	0.0217	0.0026
NSC-DLDA	25	0.0260	0.0029

The results reported in Table 6.11 suggest that DLDA applied to the full leukemia data set is the most accurate and stable of the four classifiers. In Table 6.11 the LDA (NSC-LDA), NSC and DLDA (NSC-DLDA) classifiers are all applied to the same reduced set of 25 genes in the leukemia data using $\Delta = 2.1597$. However, the DLDA classifier has a substantially lower test error and standard error than the NSC and LDA classifiers. Finally, applying the DLDA classifier to all 3 571 genes yields an average test error rate over the 200 random splits that is 20% larger than the DLDA procedure applied to the shrunken subset of 25 genes (which is a 99.30% reduction in the number of genes used).

The results summarised in Table 6.9 through 6.11 indicate that the most accurate classification procedure applied to the leukemia data set is the RBF SVM classifier with the C parameter equal to either 1 000 or 10 000 fitted to all $p = 3\,571$ genes. This procedure yields a test error rate of 0.0107 and a standard error equal to 0.0017. However, the RBF SVM classifier also achieves the worst the result of all the procedures applied to the leukemia data set. The RBF SVM classifier with $C = 1$ applied to the first 42 principal components achieved the maximum average test error rate of 0.2230 reported in Table 6.10.

As with the colon data set, Figures 6.8 through 6.10 below attempt to identify the important predictor variables in the leukemia data set, *i.e.* those which distinguish ALL from AML patients. As in the colon cancer data set, an important predictor variable is defined as a gene that is selected frequently (in at least 80% of the 200 repetitions) using the different feature extraction methods.

The following figure illustrates the selection frequencies of the genes that are selected frequently by the two-sample t-test VS procedure using p-value = 0.001 on the leukemia data set.

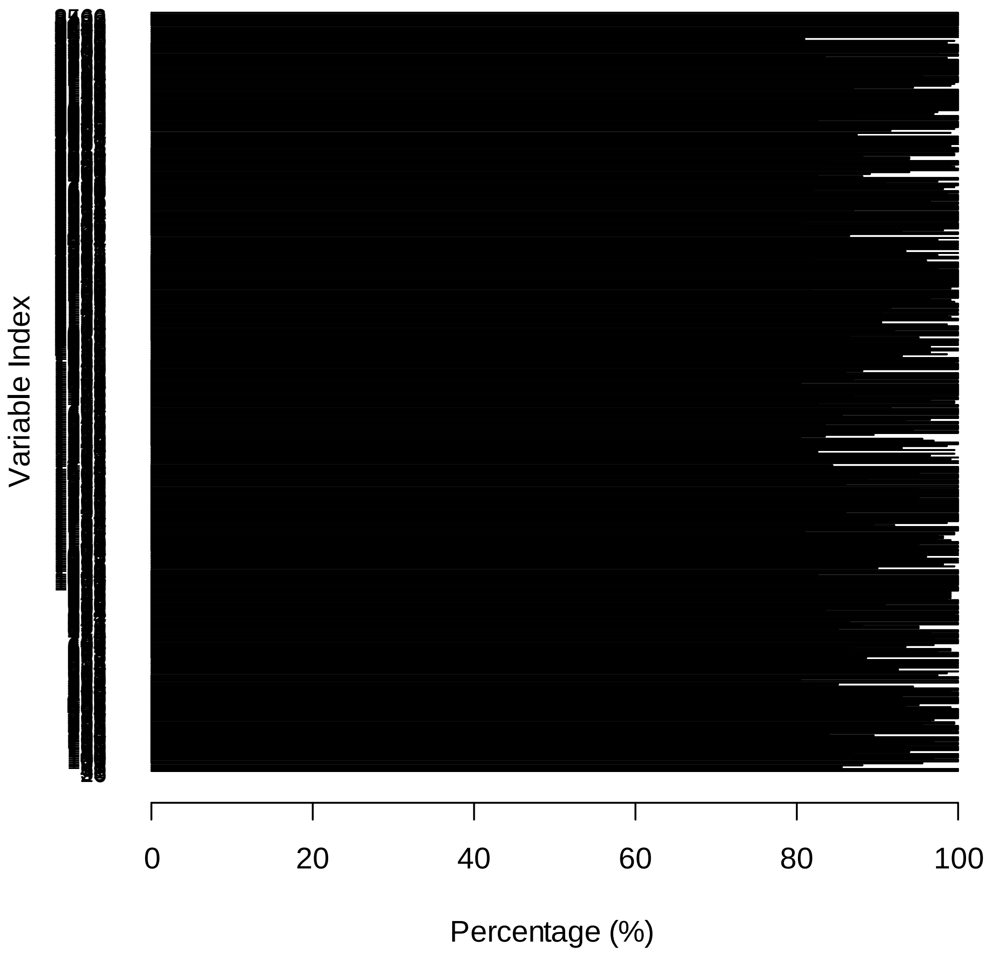

Figure 6.8: Variables selected using a two-sample t-test with a p-value of 0.001 on the leukemia data set

Of the 3 571 genes in the leukemia data set, Figure 6.8 displays 397 genes that are selected in at least 80% of the random splits. Additionally, 230 of the 397 genes displayed in Figure 6.8 are selected in 100% of the 200 repetitions. Figure 6.8 illustrates that VS on the leukemia data based on the two-sample t-test approach using the small p-value = 0.001 leads to a large number of variables being retained in the model. Note that using a p-value = 0.001 in the two-sample t-test on the colon cancer data set results in 0.70%

$\left(\frac{14}{2\,000}\right)$ of the genes being selected while the same procedure selects 11.12% $\left(\frac{397}{3\,571}\right)$ of the genes in the leukemia data set.

Additional analysis on the leukemia data set shows that the top ten genes ranked according to the two-sample t-test correspond to the indices 436, 456, 874, 956, 979, 1 099, 1 182, 1 652, 2 481 and 3 441.

Finally, although it is not shown in Figure 6.8, further analysis of the results revealed that 61.55% of the 3 571 genes in the leukemia data set are never selected in the 200 random splits and thus can be considered irrelevant in distinguishing AML from ALL patients.

The figure below displays the indices of the genes selected in at least 80% of the 200 random splits of the leukemia data set based on correlation thresholding applied at the two best values, namely 0.1000 and 0.2469.

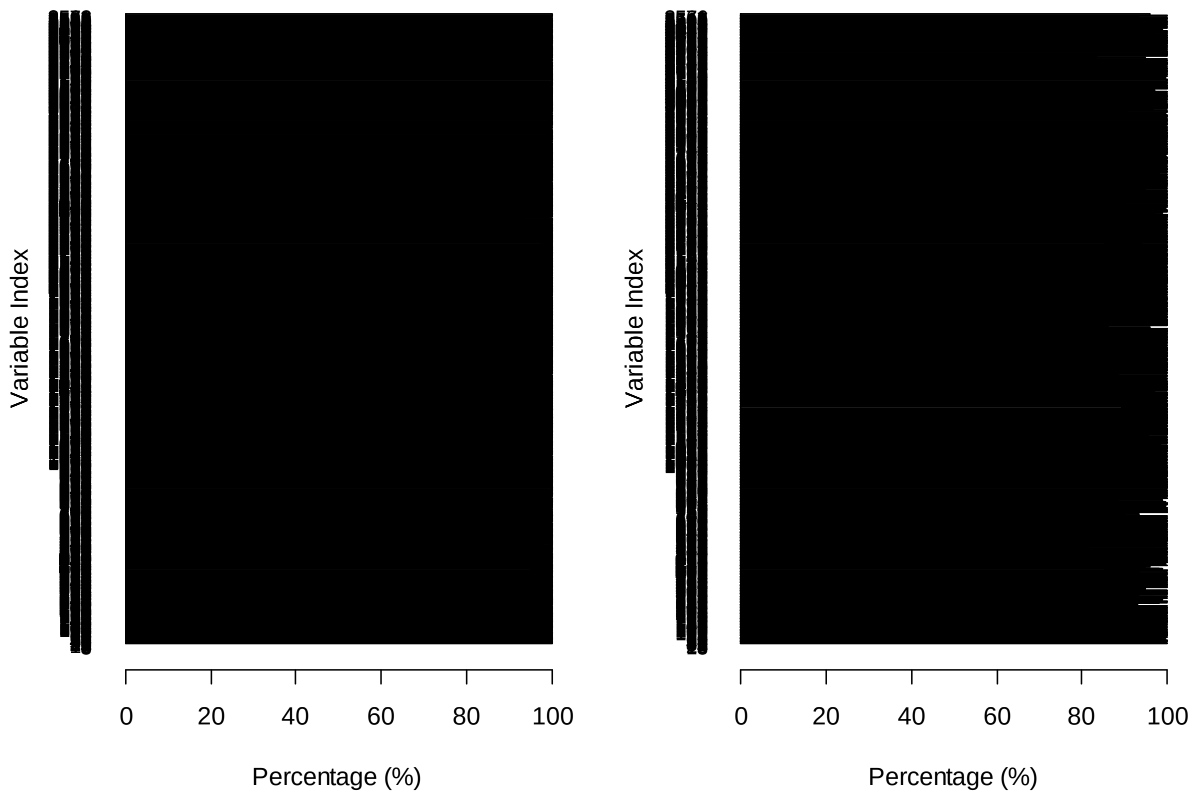

Figure 6.9: Variables selected using correlation thresholding at a value of 0.1000 (left) and 0.2469 (right) on the leukemia data set

As with Figure 6.5 for the colon data set, the small correlation thresholding values used on the leukemia data set leads to a large number of variables being selected. Consequently Figure 6.9 is very difficult to interpret as 2 125 genes are selected at least

80% of the time using the correlation value of 0.1000 (left panel) and 1 058 genes are selected at least 80% of the time using the correlation value of 0.2469 (right panel). In the left panel of Figure 6.9, 1 358 genes (63.91% of the genes displayed) are selected in every random split. In the right panel of Figure 6.9, 631 genes (59.64% of the genes displayed) are selected 100% of the time.

Analysis of the results not plotted in Figure 6.9 revealed that all 3 571 genes in the leukemia data set were selected in at least one of the 200 repetitions by correlation thresholding at a value of 0.1000. On the other hand, performing correlation thresholding at a value of 0.2469 on the leukemia data set results in 804 (22.51%) of the genes never being selected in the 200 random repetitions and these genes can therefore be considered irrelevant or insignificant in distinguishing between AML and ALL in leukemia patients.

As mentioned previously in Section 5.6.1, analysis of the original leukemia data set revealed that the gene with index 1 182 has the maximum absolute correlation with Y. Hence, it is reassuring that this gene is selected 100% of the time in both panels of Figure 6.9. Furthermore, in the full leukemia data set, 2 547 genes have a correlation value greater than 0.1000 in absolute value and 1 336 genes have a correlation value greater than 0.2469 in absolute value.

Figure 6.10 illustrates the frequencies of the genes that were selected by the NSC procedure using $\Delta_{opt} = 2.1597$ for each of the 200 splits. Only the genes that appear more than 80% of the time are displayed.

Out of the 16 genes displayed in Figure 6.10 eleven were selected by the NSC classifier 100% of the time. Although not shown in Figure 6.10, it is interesting to note that 3 465 (97.03%) of the 3 571 genes in the leukemia data set are never selected when using the NSC procedure.

It should be noted that the 16 genes displayed in Figure 6.10 are selected 100% of the time in Figure 6.8 and 6.9. From the results shown in Figure 6.8 through 6.10 it appears that the 16 genes from the leukemia data presented in Figure 6.10 are especially significant in distinguishing between AML and ALL as they are selected at least 80% of the time in all three of the variable selection methods.

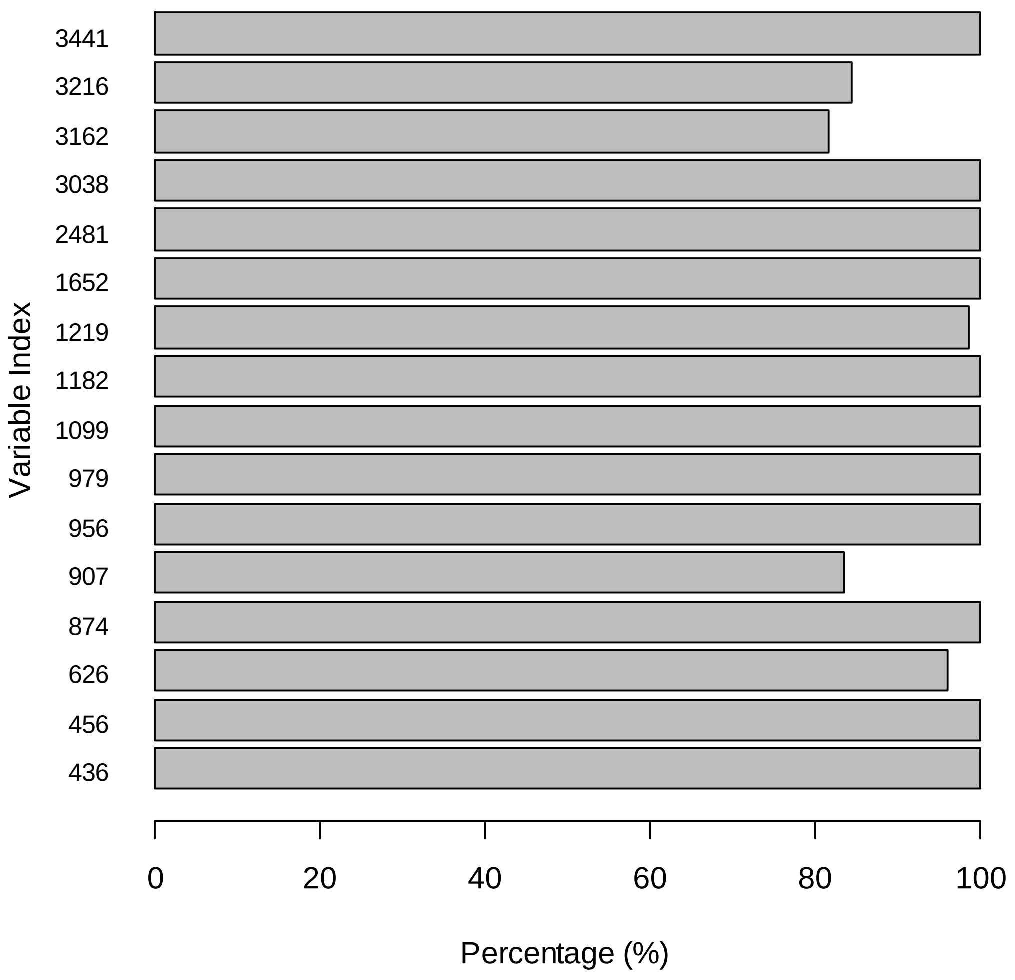

Figure 6.10: Variables selected using the NSC procedure with $\Delta_{opt} = 2.1592$ on the leukemia data set

To conclude, the SVM classifier with C =10 000 applied to all the genes in the leukemia data set is the best classification procedure considered in the thesis, achieving the minimum test error rate of 0.0107 over the 200 random splits.

6.2.3 Results on the first simulated data set

The following three tables summarise the results of the classification procedures applied to the first simulated data set. The FNR is not computed for Sim1, as it is not specified whether or not one population represents normal tissue. The optimal threshold values used are those reported previously in Section 5.6. The NSC Δ shrinkage parameter used in the KNN, SVM and NSC classifiers is 0.5013 (note that this is the optimal value determined using the NSC classifier reported in Table 5.15). Using the results reported in Table 5.5, the optimal correlation thresholding value was set to 0.3350 for both the

KNN and SVM classifiers on Sim1. However, the results in Table 5.5 indicate that the optimal p-value in the two-sample t-test VS procedure differs for the KNN and the RBF SVM classifiers. For the KNN classifier the optimal p-value $= 0.01$ while for the RBF SVM classifier the p-value $= 0.05$ was used in the two-sample t-tests.

Table 6.12 below provides the results of the KNN classification method for $K \in \{1, 3, 5\}$ on the first simulated data set.

Table 6.12: Results of the KNN classification procedures on Sim1

Method	$K = 1$			$K = 3$			$K = 5$		
	No. of feature	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**
KNN	1 000	0.3869	0.0113	1 000	0.3488	0.0103	1 000	0.2944	0.0099
fastKNN	2	0.5431	0.0130	6	0.4594	0.0107	10	0.3819	0.0104
ttest-KNN	36	0.2763	0.0102	36	0.2775	0.0107	36	0.2600	0.0099
cor-KNN	74	0.2456	0.0102	74	0.2350	0.0090	74	0.2175	0.0085
cor-fastKNN	2	0.2631	0.0103	6	0.2300	0.0092	10	0.2044	0.0084
PCA-KNN	27	0.4394	0.0093	27	0.4238	0.0094	27	0.3838	0.0102
ttest-SPCA-KNN	17	0.2694	0.0101	17	0.2575	0.0095	17	0.2669	0.0096
cor-SPCA-KNN	21	0.2388	0.0098	21	0.2138	0.0093	21	0.2094	0.0087
NSC-KNN	128	0.2600	0.0101	128	0.2338	0.0092	128	0.2288	0.0099
Average		0.3247	0.0105		0.2977	0.0097		0.2719	0.0095

The average test error rate over the 200 random splits of the Sim1 data set reported in Table 6.12 ranges from 0.2044 to 0.5431. The standard errors of the KNN procedures on Sim1 follow the theoretical pattern and the average over the nine KNN procedures in Table 6.12 decreases as K increases. The most accurate and stable KNN classification procedure in Table 6.12 is cor-fastKNN with $K = 5$ achieving the minimum test error rate of 0.2044 and the minimum standard error of 0.0084. Looking at the 1-NN procedures in Table 6.12, cor-SPCA-KNN yields the minimum test error rate of 0.2388. For $K = 3$, cor-SPCA-KNN is also the most accurate procedure achieving an average test error rate of 0.2138. Note that except for the ttest-KNN procedure, the nine KNN classification procedures' average test error rates decrease as K increases from 1 to 3. In Table 6.12 the cor-SPCA-KNN procedures for $K \in \{1, 3, 5\}$ all apply the KNN classifier to the same reduced set of 21 features (using the correlation threshold value of 0.3550), yet the accuracy of the KNN classifier in terms of average test error rate improves as the value of K increases from 1 to 5.

Looking at the 5-NN classifier in Table 6.12, cor-fastKNN has the minimum average test error rate of 0.2044, followed by cor-SPCA-KNN yielding the second lowest average test error rate of 0.2094. It should be noted that this is the first application in which the weighted voting rule used in fastKNN outperforms the majority voting rule in ordinary KNN. The cor-fastKNN procedure (for $K = 5$) makes use of ten features and achieves an average test error rate that is 6.03% lower than that of the ordinary KNN classifier applied to 74 genes obtained using correlation thresholding. As expected, the fastKNN procedure with $K = 1$ performs poorly on Sim1, achieving a test error rate of 0.5431, which is worse than the test error achieved from random guessing.

Note that although the 1-NN classifier applied to the first 27 PCs computed on Sim1 yields a test error rate that is nearly double the minimum test error rate in Table 6.12, it achieves the minimum standard error of 0.0093, indicating that its classification accuracy is stable over the 200 random splits.

The last row of Table 6.12 shows that on average the classification accuracy and stability of the nine KNN classifier improve as the number of nearest neighbours considered increases from 1 to 3 to 5. Finally, from Table 6.12 it can be concluded that the best KNN procedure is cor-fastKNN with $K = 5$.

The results for the eight SVM classification procedures for $C \in \{1, 1\ 000, 10\ 000\}$ applied to the Sim1 data set are summarised in Table 6.13. Note that for the fastKNN feature extraction procedure K was set to 5, resulting in 10 (5×2) new features being generated. The test error rates for the SVM classification procedures on Sim1 range from 0.1594 to 0.4475 and are smaller than those for the KNN classifier reported in Table 6.12.

The standard errors for the SVM classification procedures follow the general pattern and increase as the value of the cost parameter increases. The last row of Table 6.13 indicates that over the eight SVM classification procedures, the SVM classifier is most accurate and stable with $C = 1$.

Table 6.13: Results of the SVM classification procedures on Sim1

Method	No. of features	$C = 1$		$C = 1\,000$		$C = 10\,000$	
		Test Error	Std. Error	Test Error	Std. Error	Test Error	Std. Error
SVM	1 000	0.1875	0.0083	0.1950	0.0085	0.1944	0.0085
fastKNN-SVM	10	0.3475	0.0103	0.4475	0.0122	0.4450	0.0123
ttest-SVM	136	0.1819	0.0089	0.1850	0.0089	0.1856	0.0089
cor-SVM	74	0.1931	0.0084	0.1944	0.0084	0.1944	0.0083
PCA-SVM	27	0.1594	0.0087	0.1888	0.0096	0.1881	0.0096
ttest-SPCA-SVM	24	0.2156	0.0091	0.2169	0.0091	0.2169	0.0091
cor-SPCA-SVM	21	0.2275	0.0089	0.2275	0.0097	0.2263	0.0098
NSC-SVM	128	0.1738	0.0084	0.1738	0.0085	0.1738	0.0084
Average		0.2108	0.0089	0.2286	0.0093	0.2280	0.0094

Table 6.13 shows that the most accurate SVM classification method on Sim1 is PCA-SVM with $C = 1$, yielding an average test error rate of 0.1594. The best performing RBF SVM classification procedure with $C = 1\,000$ and 10 000 is the NSC-SVM procedure yielding an average test error rate of 0.1738, which is 9.02% larger than the best classification method in Table 6.13. It is interesting to note that for all three values of the C parameter, the NSC-SVM procedure achieves the same average test error rate of 0.1738.

Table 6.14 summarises the result of the LDA, DLDA and NSC classification methods for the Sim1 data set.

Table 6.14: Results of the LDA based classification procedures on Sim1

Method	No. of features used	Test Error Rate	Standard Error
NSC-LDA	128	0.2106	0.0092
NSC	128	0.2050	0.0088
DLDA	1 000	0.1769	0.0084
NSC-DLDA	128	0.1619	0.0083
Average		0.1886	0.0087

In Table 6.14, NSC-DLDA achieves the minimum average test error rate of 0.1619 and is also the most stable procedure. From Table 6.14 it is clear that the accuracy of the DLDA classifier improves by 8.48% when it is applied to the subset of 128 genes selected by the NSC procedure as opposed to being applied to all 1 000 genes. Interestingly, only 100 genes are known to be relevant in distinguishing between the two population groups in Sim1 and the NSC procedure reduces the number of genes from 1 000 to 128. Further analysis could be done to determine whether all 100 known relevant genes in the Sim1 data set appear in the NSC subset of 128 genes.

Figure 6.11 compares the results of the nine KNN classification procedures for $K = 5$ against the eight RBF SVM classification procedures with $C = 1$.

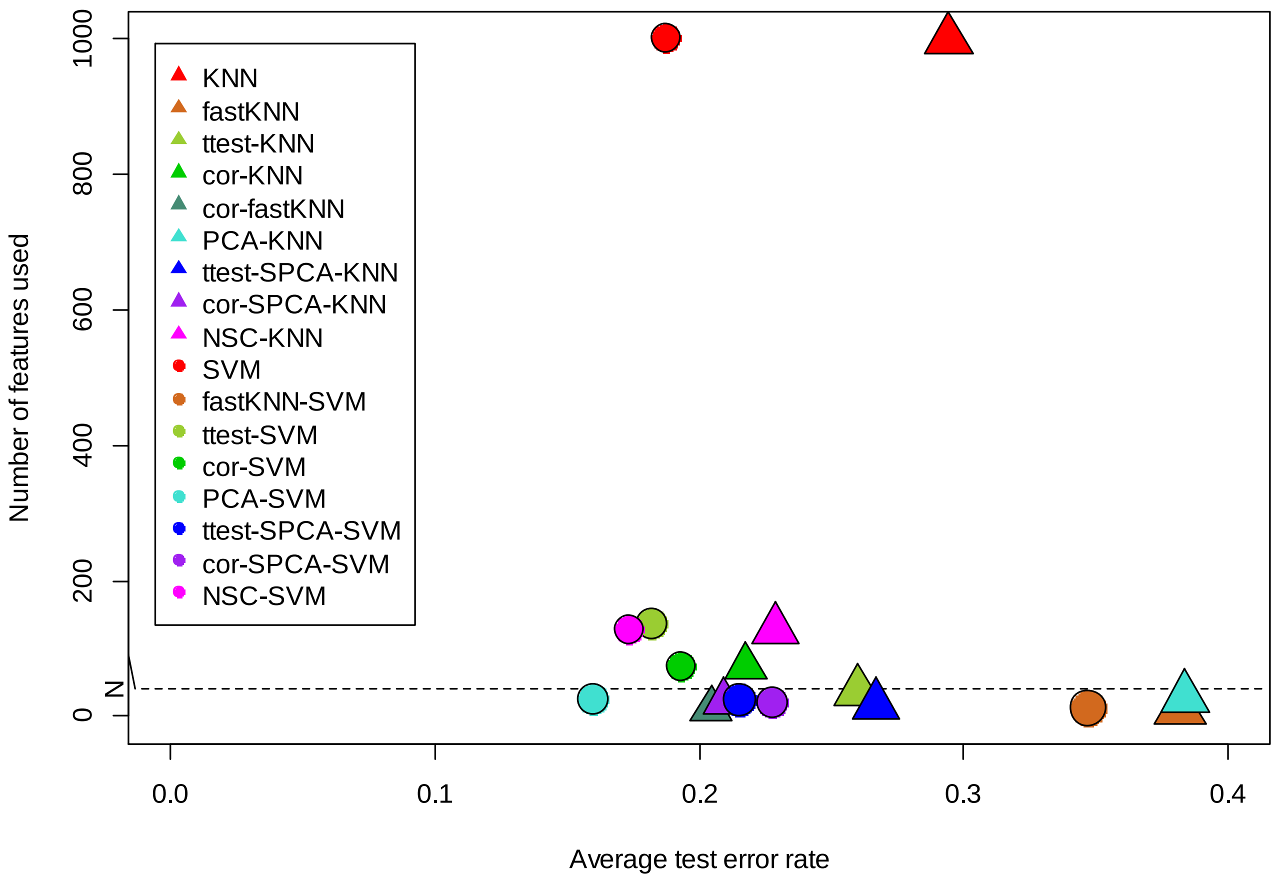

Figure 6.11: Comparison of the classification procedures on Sim1

Figure 6.11 clearly illustrates that PCA-SVM has the minimum average test error rate of the 17 classification procedures displayed (namely 0.1594) and therefore is the most accurate procedure considered on the Sim1 data set. On the other hand, PCA-KNN is the worst performing 5-NN procedure on Sim1 yielding an average test error rate of 0.3838. It is interesting that the SVM and 5-NN classifier are both applied to the same set of 27 principal components from Sim1, yet the SVM classifier achieves an average test error rate over the 200 random splits that is 58.47% lower than that of the 5-NN classifier. Furthermore, Figure 6.11 illustrates that for the eight different procedures (ignoring cor-fastKNN) the SVM classifier is superior to the 5-NN classifier.

It should be noted that the correlation coefficient, two-sample t-test and NSC procedures are all variable selection techniques that are designed to deal with scenarios where there is a location difference between the variables in two population groups such as in Sim1. Therefore, the three above mentioned techniques are expected to be superior to feature extraction procedures that are not designed to deal with a location difference scenario, such as PCA. However, this is not shown to be true for the SVM classifier in Figure 6.11 as PCA-SVM is more accurate than cor-SVM, NSC-SVM and ttest-SVM.

From the results reported in Table 6.12 through Table 6.14 it can be concluded that PCA-SVM with $C = 1$ is the most accurate classifier for the Sim1 data set.

6.2.4 Results on the second simulated data set

Several problems were encountered when applying correlation thresholding, the two-sample t-test and NSC variable selection procedures to the second simulated data set using the optimal thresholding values determined in Section 5.6. All these problems arose from selecting very strict optimal thresholding values. As a consequence, when the Sim2 data set was randomly split into training and test data sets it frequently occurred that too few or no genes were identified as important in a random split.

In Table 5.6 the largest absolute candidate correlation thresholding value considered for Sim2 (namely 0.4296) was selected as the optimal value. When correlation thresholding was applied to the full Sim2 data set at a value of 0.4296 only six genes were identified as significant. Consequently, the very strict correlation thresholding value resulted in many of the 200 random splits of Sim2 consisting of zero significant genes.

Similarly, the optimal p-value for the two-sample t-test on Sim2 is reported as 0.001 in Table 5.11. Performing a two-sample t-test on the full Sim2 data set at a p-value of 0.001 revealed that only three genes are selected as significant.

From this it can be seen that LOOCV in Section 5.6 selected very strict thresholding values for the two above-mentioned VS techniques implying that the KNN and SVM classifiers are more accurate when applied to smaller subsets of genes from Sim2. However, due to the random splitting used in determining the best classification procedure, there are often too few genes selected by the two VS procedures and therefore it is not possible to implement the KNN and SVM classifiers.

For Sim2 it is known that only the first 100 genes are relevant in distinguishing between the two groups. However, correlation thresholding incorrectly selected the following six genes as important in distinguishing between the two populations, namely the genes corresponding to the indices 382, 415, 447, 471, 473 and 408. Variable selection using the two-sample t-test at a p-value$=0.001$ also incorrectly selected the genes indexed 382, 471 and 908 as important.

The results reported in Table 5.16 suggested that the optimal NSC shrinkage parameter value using the NSC classifier is $\Delta_{opt}=0.9458$. However, there is also a problem when computing d'_{jg} as in (2.5) since there are only three genes in Sim2 that have a $|d_{jg}|$ value greater than $\Delta=0.9458$, namely the genes with indices 382, 471 and 908. Therefore, performing the NSC procedure on the full Sim2 data set (not split into training and test) using $\Delta_{opt}=0.9458$, leads to the conclusion that only the genes with indices 382, 471 and 908 are significant in distinguishing between the two groups and the remaining 997 genes in Sim2 are irrelevant and should be removed from the model. Therefore, due to the high optimal thresholding value of $\Delta_{opt}=0.9458$, the NSC VS procedure runs into problems when Sim2 is split according to the 80-20 training-test split as in many splits no genes are selected as significant. Therefore, it is not possible to create a programme that automatically selects the two most important genes if a split yielded that no genes have a $d'_{jg}>0$ since if all the $d'_{jg}=0$ it is not possible to distinguish which two are the most important.

Over and above the problems described above, it seems that the three VS methods are unsuccessful in identifying the relevant genes (those with indices from 1 to 100). This is not surprising, since Sim2 is a data set from a scenario in which there is a scale difference between the two groups. Consequently, the relevant genes identified by the three location based VS measures are purely random. Finally, it can be concluded that the t-test, correlation thresholding and NSC are not appropriate measures for performing VS in scenarios where there is a difference in scale between the two groups, such as in the Sim2 data set. Other variable selection measures should be considered in such cases, for example mutual information.

Consequently, as a result of these issues, correlation thresholding, and the two-sample t-test and NSC variable selection procedures were not applied to the Sim2 data set. Further research into applying VS procedures that test for the difference in variance between two or more groups, or more generally for different distributions, could be investigated and applied to Sim2.

The results pertaining to the KNN classification method for $K \in \{1, 3, 5\}$ on the Sim2 data set are tabulated below.

Table 6.15: Results of the KNN classification procedures on Sim2

Method	$K = 1$			$K = 3$			$K = 5$		
	No. of feature	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**
KNN	1 000	0.5500	0.0112	1000	0.4625	0.0112	1000	0.4900	0.0119
fastKNN	2	0.4831	0.0115	6	0.4456	0.0117	10	0.4313	0.0112
PCA-KNN	27	0.5244	0.0103	27	0.5331	0.0103	27	0.5331	0.0114
Average		0.5192	0.0110		0.4804	0.0111		0.4848	0.0115

The average test error rates of the KNN classifiers on Sim2 reported in Table 6.15 are very high, ranging from 0.4313 to 0.5500. Since, the test error rate of random guessing is 0.50, the results reported in Table 6.15 indicate that the KNN classifier with $K \in \{1, 3, 5\}$ is not suitable for Sim2. The most accurate KNN classifier in Table 6.15 is fastKNN with $K = 5$, yielding an average test error rate of 0.4313 over the 200 random splits of Sim2. For each value of K in the KNN classifier, fastKNN is superior to the KNN and PCA-KNN procedures. Finally, in Table 6.15 it is evident that the KNN classifier is more accurate when more NNs are considered. However, the average test error rate of the 5-NN classifiers is still as large as 0.4848.

Traditionally the KNN classifier is more stable when using a larger value of K, however in Table 6.15 the 5-NN classifier has the maximum average standard error, indicating that it has the most variation over the 200 random splits.

Table 6.16 reports the results for the three SVM classification procedures applied to the Sim2 data set. Note that for the fastKNN feature engineering procedure, K is set to 5, resulting in 10 (5×2) new features being generated.

Table 6.16: Results of the SVM classification procedures on Sim2

Method	No. of features	$C = 1$		$C = 1\,000$		$C = 10\,000$	
		Test Error	Std. Error	Test Error	Std. Error	Test Error	Std. Error
SVM	1000	0.3994	0.0092	0.4331	0.0110	0.4331	0.0110
fastKNN-SVM	10	0.4356	0.0099	0.4725	0.0110	0.4725	0.0115
PCA-SVM	27	0.4869	0.0106	0.4688	0.0108	0.4688	0.0108
Average		0.4406	0.0099	0.4581	0.0109	0.4581	0.0111

The best procedure in Table 6.16 is the SVM classifier with $C = 1$ fit to all the genes in the Sim2 data set, yielding the minimum test error rate of 0.3394. In Table 6.16, it is evident that for $C \in \{1, 1\,000, 10\,000\}$ the RBF SVM classifier is more accurate when fit to all 1 000 genes in Sim2 than the subset of features (which are linear combinations of all the genes) from fastKNN feature engineering or PCA.

The standard errors of the SVM classification procedures in Table 6.16 follow the general pattern and increases as the value of C increases.

Table 6.17 reports the results for the DLDA classifier applied to all the genes in Sim2.

Table 6.17: Results of the DLDA classifier on Sim2

Method	No. of features used	Test Error Rate	Standard Error
DLDA	1000	0.4275	0.0108

The DLDA classifier does not perform well on Sim2, yielding an average test error rate of 0.4275 over the 200 random splits.

From the results reported in Table 6.15 through 6.17, it is clear that none of the three base classifiers considered perform well on either the full or the reduced version of the Sim2 data set. Nevertheless, of all the classification procedures considered, the RBF SVM classifier with $C = 1$ applied to all 1 000 genes achieved the smallest test error rate of 0.3994.

6.2.5 Results on the third simulated data set

The tables below report the results of the classification procedures applied to the third simulated data set using the optimal threshold values previously reported in Section 5.6.

The NSC Δ shrinkage parameter used in the KNN, SVM and NSC classifiers was 0.7298, the optimal value determined using the NSC classifier reported in Table 5.17.

The optimal p-value $= 0.01$ was used in the two-sample t-test VS procedure for both the KNN and SVM classifiers. However, the optimal correlation thresholding value used differed for the KNN and the RBF SVM classifiers. For the 1-NN and 3-NN classifiers the correlation thresholding value was set to 0.3845, while for the 5-NN classifier and the RBF SVM classifier the correlation thresholding value was set to 0.1000.

Table 6.18: Results of the KNN classification procedures on Sim3

Method	$K = 1$			$K = 3$			$K = 5$		
	No. of feature	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**
KNN	1 000	0.3350	0.0097	1 000	0.2731	0.0093	1 000	0.2494	0.0099
fastKNN	2	0.4144	0.0122	6	0.3356	0.0117	10	0.2869	0.0107
ttest-KNN	45	0.3144	0.0106	45	0.2700	0.0095	45	0.2638	0.0089
cor-KNN	57	0.2819	0.0105	57	0.2563	0.0093	642	0.2256	0.0098
cor-fastKNN	2	0.3106	0.0108	6	0.2813	0.0103	10	0.2900	0.0101
PCA-KNN	27	0.4206	0.0109	27	0.3900	0.0096	27	0.3338	0.0094
ttest-SPCA-KNN	18	0.2994	0.0097	18	0.2669	0.0089	18	0.2506	0.0091
cor-SPCA-KNN	20	0.2744	0.0095	20	0.2556	0.0096	27	0.2444	0.0094
NSC-KNN	36	0.2906	0.0109	36	0.2550	0.0093	36	0.2413	0.0081
Average		0.3268	0.0105		0.2871	0.0097		0.2651	0.0095

The results for the Sim3 data set summarised in Table 6.18 show that both the average test error rate and standard error of the nine extensions of the KNN classifier decrease as the value of the number of nearest neighbours considered in the classifier increases from 1 to 3 and then to 5. This is similar to the results reported in Table 6.12 with the KNN classifiers applied to Sim1.

The minimum average test error rate in Table 6.18 is 0.2256, achieved by the 5-NN classifier applied to the data set consisting of only 57 genes after undergoing variable selection using correlation thresholding (cor-KNN). It is also the least volatile of the KNN procedures in Table 6.18 over the 200 random repetitions, having the minimum standard error of 0.0081.

Considering the 1-NN classifier in Table 6.18, the cor-SPCA-KNN procedure is the most accurate, yielding an average test error rate of 0.2744. The 3-NN classifier is most accurate when applied to the NSC reduced subset of 36 genes.

The results for the eight RBF SVM classification procedures applied to Sim3 appear in Table 6.19.

Table 6.19: Results of the SVM classification procedures on Sim3

Method	**No. of features**	$C = 1$		$C = 1\ 000$		$C = 10\ 000$	
		Test Error	**Std. Error**	**Test Error**	**Std. Error**	**Test Error**	**Std. Error**
SVM	1 000	0.1919	0.0080	0.2094	0.0092	0.2094	0.0092
fastKNN-SVM	10	0.3100	0.0106	0.3313	0.0110	0.3313	0.0109
ttest-SVM	45	0.2825	0.0086	0.2963	0.0090	0.2950	0.0090
cor-SVM	642	0.2106	0.0088	0.2225	0.0091	0.2225	0.0091
PCA-SVM	27	0.1756	0.0084	0.1775	0.0087	0.1781	0.0087
ttest-SPCA-SVM	18	0.2825	0.0091	0.3106	0.0094	0.3100	0.0094
cor-SPCA-SVM	27	0.2044	0.0095	0.2038	0.0090	0.2044	0.0091
NSC-SVM	36	0.2663	0.0090	0.2900	0.0101	0.2913	0.0100
Average		0.2405	0.0090	0.2552	0.0094	0.2552	0.0094

The most accurate SVM classification method on the Sim3 data set reported in Table 6.19 is the PCA-SVM procedure with $C = 1$, yielding an average test error rate of 0.1756. PCA-SVM with $C = 1$ is also the most stable procedure over the 200 repetitions with a standard error of 0.0080. (Note that the PCA-SVM with C =1 was also the most accurate SVM procedure on Sim1). The most accurate RBF SVM classifiers for C =1 000 and 10 000 are also the PCA-SVM procedures with average test error rates of 0.1775 and 0.1781 respectively. It is interesting that the SVM classifiers perform well on linear combinations of the original genes in Sim3 using PCA, while the SVM classifier performed poorly on the linear combination of genes using PCA on the leukemia data set.

The results reported in Table 6.19 indicate that the subset of genes extracted by fastKNN with $K = 5$ is the worst dimension reduction technique for the RBF SVM classifier on the Sim3 data set. From the last row in Table 6.19 it can be observed that on average the test error rate of the eight SVM classifiers on the Sim3 data set increases (accuracy worsens) as the value of the C parameter in the RBF SVM classifier increases from 1 to 1 000 and to 10 000.

The results of the LDA, NSC and DLDA classification procedures on the 200 random splits of Sim3 are provided in the table below.

Table 6.20: Results of the LDA based classification procedures on Sim3

Method	**No. of Features used**	**Test Error Rate**	**Standard Error**
NSC-LDA	36	0.4475	0.0129
NSC	36	0.2681	0.0080
DLDA	1 000	0.1950	0.0085
NSC-DLDA	36	0.2394	0.0083
Average		0.2875	0.0094

It is evident that of the four LDA based classification procedures presented in Table 6.20 the DLDA classifier fit to the full Sim3 data sets achieves the minimum average test error rate of 0.1950. The remaining three classification procedures are applied to a reduced set of 36 genes from the Sim3 data set. The poor performance of NSC-LDA in Table 6.20 could be as a result of the LDA classifier using the assumption that the two populations have the same variance-covariance matrix which is incorrect for the Sim3 data set.

Figure 6.12 displays the results of the nine 5-NN classification procedures and the eight SVM classification procedures with $C = 1$ on the Sim3 data set.

Figure 6.12 reinforces the conclusion that for the eight different classification procedures, the SVM classifier with $C = 1$ is superior to the 5-NN classifiers.

In Figure 6.12 it can be observed that for the 5-NN classifier, the cor-KNN procedure is the optimal procedure as it achieves the minimum test error rate of the nine 5-NN classification procedures. Cor-KNN yields a test error rate of 0.2256 and applies the 5-NN classifier to the reduced set of 642 genes obtained using correlation thresholding. Note that for cor-KNN the number of features used is larger than $N = 100$.

Figure 6.12 clearly illustrates that when applied to the reduced set of 642 genes obtained using correlation thresholding, the KNN classifier is superior to the fastKNN classification procedure which uses feature engineering to further reduce the 642 genes to two features and then classifies according to the weighted voting rule.

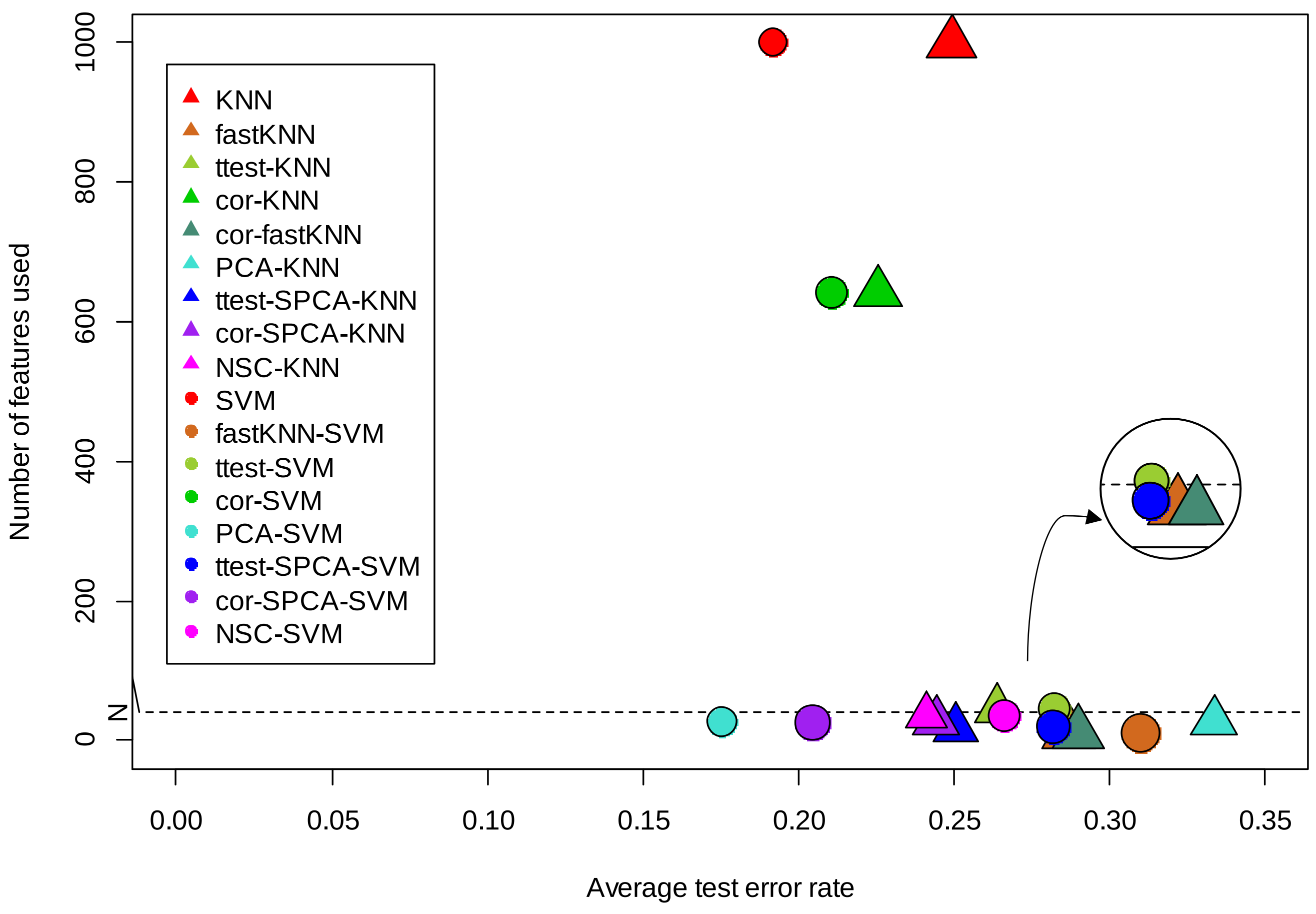

Figure 6.12: Comparison of the 5-NN and SVM classification procedures on Sim3

Finally, in Figure 6.12, the sizes of the 17 points represent the standard errors of the classification procedures over the 200 random splits of Sim3. The standard errors have been magnified by 300 points in R. Therefore, it is clear that the fastKNN and fastKNN-SVM procedures (shown in orange) vary the most among the 200 random splits.

In conclusion, the results summarised in Tables 6.18 through Table 6.20 suggest that on average the RBF SVM classifiers and in particular PCA-SVM with $C = 1$ is the most accurate and stable of the base classifiers applied to Sim3. Overall the KNN classification procedures reported in Table 6.18 have the worst performance on Sim3.

6.2.6 Results on the SRBCT data set

Tables 6.21 and 6.22 summarise the results of the different classification procedures applied to the SRBCT data set. For the SRBCT data set, the optimal thresholding values applied in the VS techniques were those determined using LOOCV in Section 5.6.4. Correlation thresholding was implemented at a value of 0.7253, and NSC VS was implemented using $\Delta_{opt} = 3.3309$ for both the KNN and NSC classifiers. As in the binary data sets the number of principal components and supervised principal components retained in the SRBCT data set was determined by setting $\omega_{frac} = 0.9$. Five VS and dimension reduction techniques were applied to the SRBCT data set followed by the KNN and NSC classifiers, thereby resulting in seven versions of the KNN based classifier and one NSC classifier procedurre. The test error rates were computed for each classifier. This was repeated 200 times and the average number of features used, the average test error and standard error were calculated for the eight classifiers.

Performing random guessing on the SRBCT data set yields a test error rate of 75%. The FNR was not calculated for the SRBCT data set as it is a multi-class data set.

The results for the KNN classification procedures on the SRBCT data set are reported in the table below for $K \in \{1, 3, 5\}$.

Table 6.21: Results of the KNN classification procedures on the SRBCT data set

Method	**$K = 1$**			**$K = 3$**			**$K = 5$**		
	No. of feature	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**
KNN	2 308	0.1150	0.0050	2 308	0.0933	0.0045	2 308	0.1039	0.0049
fastKNN	4	0.1742	0.0059	12	0.1464	0.0049	20	0.1111	0.0046
cor-KNN	7	0.2722	0.0072	7	0.3053	0.0066	7	0.3144	0.0066
cor-fastKNN	4	0.3503	0.0073	12	0.3453	0.0071	20	0.3475	0.0069
PCA-KNN	35	0.1383	0.0058	35	0.1064	0.0045	35	0.1300	0.0053
cor-SPCA-KNN	4	0.3125	0.0079	4	0.3258	0.0078	4	0.3306	0.0067
NSC-KNN	48	0.0006	0.0004	48	0.0025	0.0009	48	0.0008	0.0005
Average		0.1947	0.0056		0.1893	0.0052		0.1912	0.0051

The results summarised in Table 6.21 indicate that for $K \in \{1, 3, 5\}$ in the KNN classifier, the most accurate of the seven KNN classifiers on the SRBCT data set is the NSC-KNN procedure. This procedure entails implementing NSC as a VS method using $\Delta_{opt} = 3.3309$ and then applying the KNN classifier on the resulting subset of 48 genes. Furthermore, NSC-KNN also achieves the minimum standard error in Table 6.21 indicating that there is very little variation in the performance of NSC-KNN across the 200 random splits. Overall the minimum test error rate averaged over the 200 random splits in Table 6.21 is 0.0006, which is achieved when the NSC VS procedure was used to reduce the SBRCT data set before applying the 1-NN classifier.

The standard errors reported in Table 6.21 vary from 0.0004 to 0.0079. The standard errors averaged over the seven KNN classification procedures for each value of K follows the theoretical trend and decreases as the value of K increases.

The results given in Table 6.21 indicate that the fastKNN classifier applied to the reduced data set using correlation thresholding at a value of 0.7253 achieves the maximum average test error rate out of the seven KNN methods for all values of K. As expected, the fastKNN classification procedure improves in terms of both classification accuracy and consistency (smaller standard error) as the value of K increases.

Note that applying the NSC procedure with $\Delta_{opt} = 3.3309$ results in a 97.92% decrease in the number of genes used, from 2 308 to 48. In addition, applying the KNN classifier to the reduced data set instead of the full SRBCT data set results in a 99.48% decrease in the average test error rate.

The results of the NSC classifier on the SRBCT data set are summarised in Table 6.22.

Table 6.22: Results of the NSC classifier on the SRBCT data set

Method	No. of features used	Test Error Rate	Standard Error
NSC	48	0.0003	0.0003

It is clear from Table 6.21 and 6.22 that the NSC classifier is the most accurate classification procedure considered on the SRBCT data set - it achieves both a test and standard error of 0.0003 when the number of genes used is shrunk from 2 308 to 48. Further analysis of the results indicate that over the 200 random training-test splits there was only a single misclassification in one split using the NSC classifier.

Figure 6.13 graphically presents the results for the KNN classifier with $K = 1$ (given in Table 6.21) and the NSC classifier (given in Table 6.22) for the SRBCT data set. Although on average the 3-NN classification procedures have the minimum average test error rate over the seven KNN classifiers in Table 6.21, the 1-NN classification procedures are plotted since NSC-KNN with $K = 1$ achieves the overall minimum test error rate in Table 6.21.

The size of the points in Figure 6.13 represents the standard errors of the classifiers. The largest point plotted in Figure 6.13 belongs to cor-SPCA-KNN which illustrates that it has the maximum standard error over the 200 random splits, at a value of 0.0079.

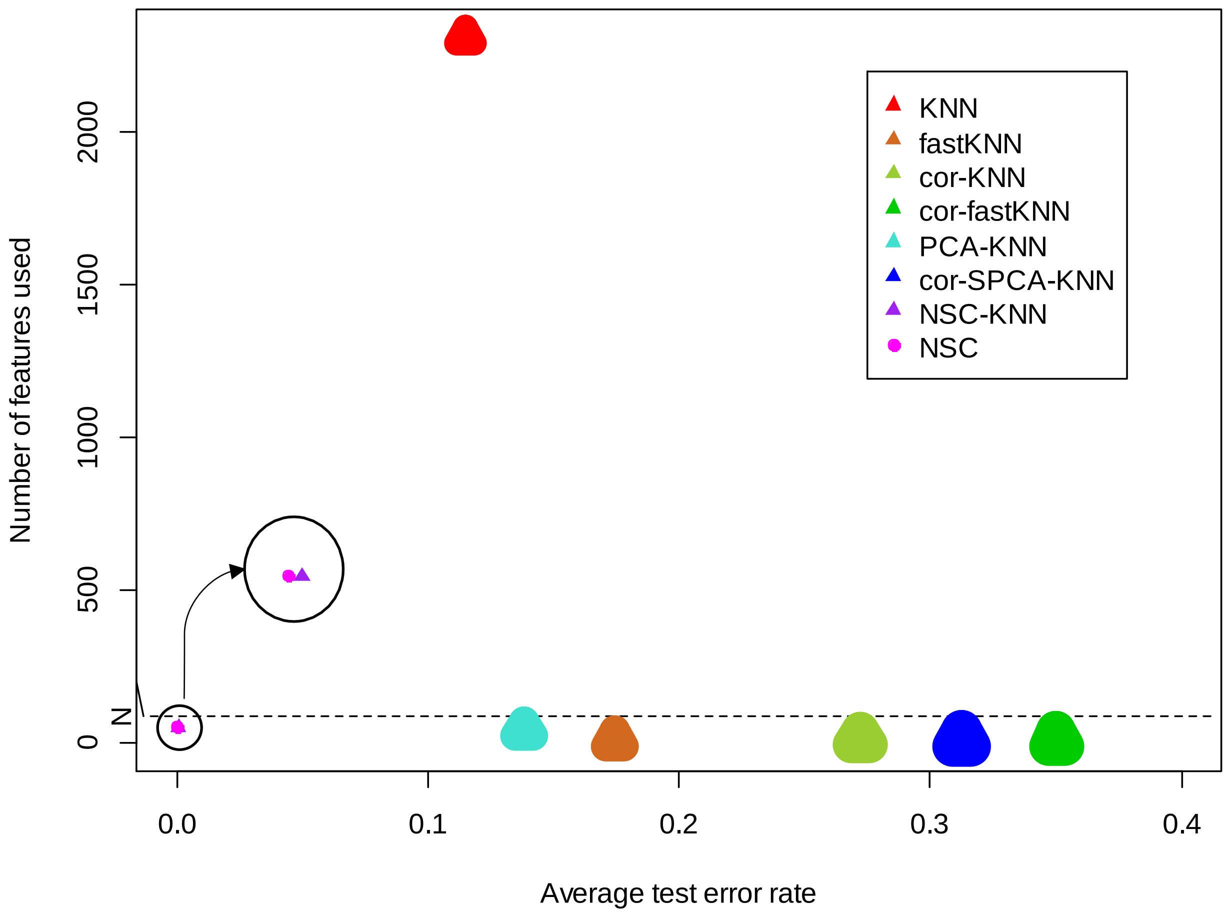

Figure 6.13: Comparison of the 1-NN and NSC procedures on the SRBCT data set

Figure 6.13 confirms the results in Table 6.22 and 6.21 that the NSC classifier and 1-NN classifier are both applied to the same subset of 48 genes. However, the NSC classifier's average test error rate is half of the average test error rate achieved by the 1-NN classifier. In Figure 6.13, the arrow pointing to the two points shows zoomed in versions of the NSC-KNN and NSC classifiers on a larger scale in order to illustrate that the NSC classifier has a lower average test error rate than the NSC-KNN classifier.

The horizontal dotted line in Figure 6.13 indicates where $p = N$, and therefore it is clear that only KNN applied to the full SRBCT data set faces the curse of high-dimensionality, although cor-SPCA-KNN still makes use of all 2 308 genes in the four supervised principal components.

It is clear in Figure 6.13 that the cor-fastKNN procedure has the highest average test error rate of the eight classifiers displayed in the plot. Therefore, from the results reported in Table 6.21 and 6.22 and represented in Figure 6.13, it can be concluded that the NSC classifier is the most accurate classification procedure when applied to the SRBCT data set.

The following figure illustrates the selection frequencies of the genes in the SBRCT data set for each of the 200 random splits after implementing correlation thresholding at a value of 0.7253 and the NSC VS procedure with $\Delta_{opt} = 3.3309$. Figure 6.14 only displays the genes that appear in at least 80% of the 200 random splits of the SRBCT data set.

Note that the results reported in Table 6.21 indicate that the modal number of genes used in the correlation thresholding and NSC VS procedure on the SRBCT data set are 7 and 48 respectively.

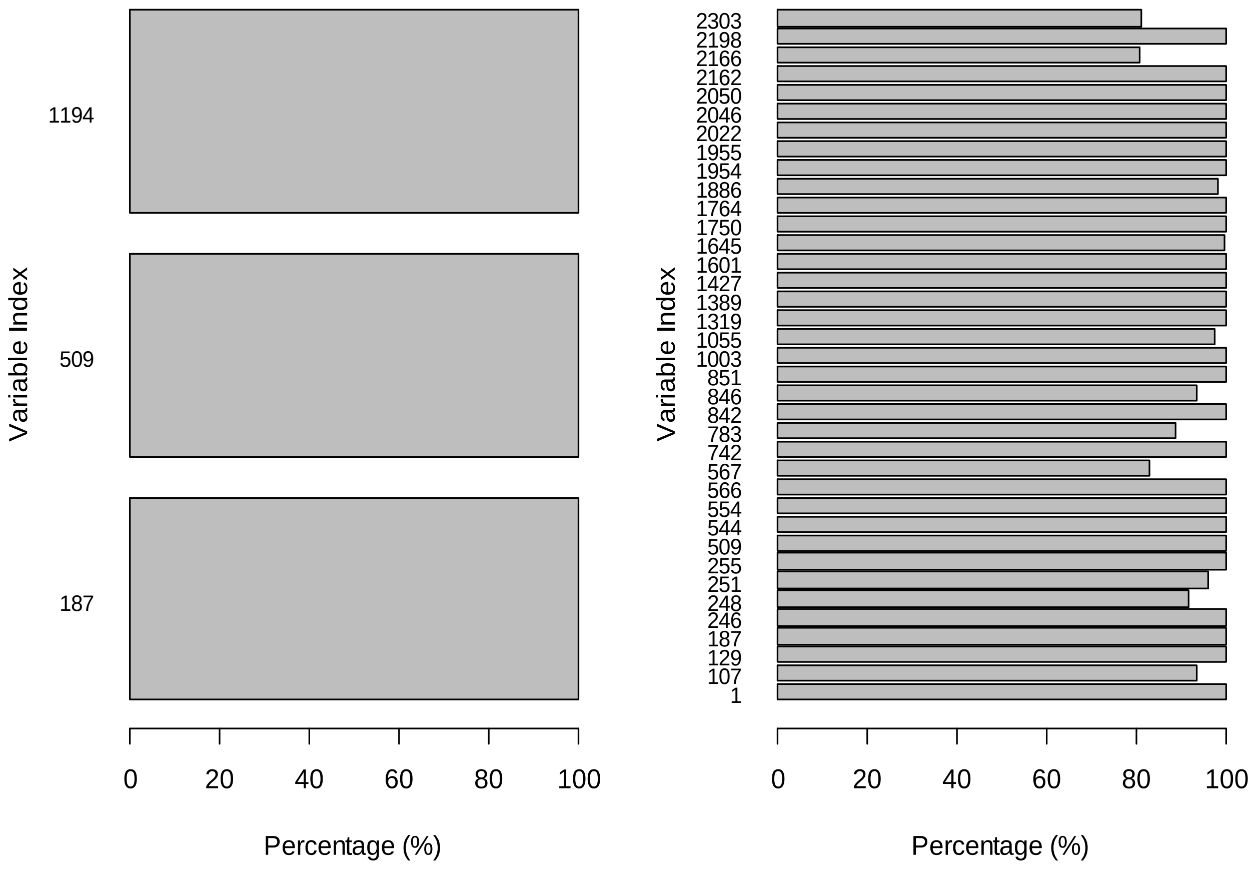

Figure 6.14: Variables selected using correlation thresholding at 0.7253 (left) and the NSC procedure with Δ_{opt}=3.3309 (right) on the SRBCT data set

The left panel in Figure 6.14 plots the three genes that are selected at least 80% of the time, all of which are in fact selected in 100% of the random splits on the SRBCT data set. Implementing correlation thresholding at a value of 0.7253 on the SRBCT data set resulted in 2 293 (99.35%) of the $p = 2\ 308$ possible genes never being selected in the 200 random splits. Note that 0.7253 is the largest correlation threshold value considered in Table 5.8, which explains why so few genes are identified as significant.

Turning attention to the right panel of Figure 6.14, of the 37 genes displayed 26 are selected 100% of the time which is approximately 70.27% of the genes displayed. Furthermore, the NSC VS procedure suggests that 2 198 (95.23%) of all the genes in the SRBCT data set are irrelevant and insignificant in distinguishing between the four SRBCT populations groups as they are never selected in the 200 repetitions.

Interestingly, in the NSC VS procedure 1.13% (26) of the genes appear 100% of the time in the random splits while only 0.13% appear every time when using the correlation thresholding on the SRBCT data set. In Figure 6.14 it can be seen that only genes 187 and 509 in the SRBCT data set are selected in every random split by both VS techniques. However, gene 1 194 was selected 100% of the time by the correlation thresholding but was selected by NSC less than 80% as it is not displayed in the right panel of Figure 6.14.

Note that calculations from Table 5.18 yielded that the genes with indices 1 194, 187, 509, 2 046 and 1 003 (in decreasing order) all have a correlation with Y above the thresholding value of 0.7253.

Since the NSC classifier is the optimal classification procedure on the SRBCT data set, the following table displays the variable indices of the 37 genes in the SRBCT data set that were identified in over 80% of the splits by the NSC VS procedure (displayed in the right of Figure 6.14) and their respective image clone identifier number and the gene expression name.

Table 6.23: Description of the important gene expressions in the SRBCT data set

Gene Index	Image Clone ID	Gene Expression (Abbreviation)	Freq. (%)
1	21 652	Alpha 1 catenin (cadherin-associated protein) (CTNNA1)	100.00
107	365 826	Growth arrest-specific protein 1 (GAS1)	93.50
129	298 062	Troponin T2, cardiac muscle isoforms(TNNT2)	100.00
187	296 448	insulin-like growth factor 2 (somatomedin A) (IGF2)	100.00
246	377 461	Caveolin 1 (caveolae protein) (CAV1)	100.00
248	897164	catenin (cadherin-associated protein), alpha 1, (CTNNA1)	91.50
251	486787	calponin 3, acidic (CNN3)	96.00
255	325 182	N-cadherin (neuronal) (CDH2)	100.00
509	207 274	insulin-like growth factor 2 (IGF2)	100.00
544	1 416 782	creatine kinase, brain (CKB)	100.00
554	461 425	Myosin MYL4 (MYL4)	100.00
566	357 031	Tumor necrosis factor tansporter, alpha chain (TNFAIP6)	100.00

Gene Index	Image Clone ID	Gene Expression (Abbreviation)	Freq. (%)
567	768370	TIMP3	83.00
742	812 105	ALL1-fused gene from chromosome 1q (AF1Q)	100.00
783	767183	hematopoietic cell-specific Lyn substrate 1 (HLCS1)	88.50
842	810 057	Cold shock domain protein A	100.00
846	183 337	major histocompatibility complex, class II, DM alpha (HLA-DMA)	93.50
851	563 673	Antiquitin 1 (ATQ1)	100.00
1 003	796 258	Sarcoglycan alpha (dystrophin-associated glycoprotein) [SGCA]	100.00
1 055	1 409 509	Troponin T1, slow skeletal muscel isoforms (TNNT1)	97.50
1 319	866 702	Fas-associated protein tyrosine phosphatase 1 (PTPN13)	100.00
1 389	770 394	IgG Fc fragment receptor transported, alpha chain (FCGRT)	100.00
1 427	504 791	Glutathione S-transferase A4 (GSTA4)	100.00
1 601	629 896	microtubule-associated protein 1B (MAP1B)	100.00
1 645	52076	neuroblastoma protein (NOE1)	99.50
1 750	233 721	insulin-like growth factor binding protein 2 (IGFBP2)	100.00
1 764	44 563	Growth associated protein 43 (GAP43)	100.00
1 886	897 788	Receptor type protein tyrosine phosphatase F (PTPRF)	98.00
1 954	814 260	Follicular lymphoma variant translocation protein 1 (FVT1)	100.00
1 955	784 224	Fibroblast growth factor receptor 4 (FGFR4)	100.00
2 022	204 545	EST (EST)	100.00
2 046	244 618	EST	100.00
2 050	295 985	EST (EST)	100.00
2 162	308 163	EST (EST)	100.00
2 166	296616	Homo sapiens cDNA FLJ35702 fis	80.50
2 198	212 542	cDNA DKFZp586J2118 (EST)	100.00
2 303	782 503	Homo sapiens clone 23716 mRNA sequence (EST)	81.00

The gene with index 2 166 is selected the minimum number of times of all the 37 genes in Table 6.23 - it is selected in 161 of the 200 random splits of the SRBCT data set. Of the 37 genes reported in Table 6.23 only genes 248 and 2 166 in the SRBCT data set do not appear in either the subset of 43 genes selected by NSC in the paper by Tibshirani *et al.* (2001) and the 96 genes from the neural network method of Khan *et al.* (2001).

In the paper by Tibshirani *et al.* (2001), the NSC procedure identifies a subset of 43 genes having at least one non-zero difference between the groups. In Table 6.23, 29 genes (78.38% of the genes displayed) appear in the subset of 43 genes selected by NSC in Tibshirani *et al.* (2001). The eight variables displayed in Table 6.23 that do not appear in the subset of 43 genes are the variables with indices 248, 251, 544, 567, 783, 1 061, 2 166 and 2 303. Note that the variables with indices 251, 544, 567, 783, 1 601 and 2 303 are in the subset from the neural network method of *Khan et al.* (2001), but not the NSC subset of Tibshirani *et al.* (2001). The variable with index 842 is in the NSC subset of genes from Tibshirani *et al.* (2001), but not in the neural network method of Khan *et al.* (2001). Therefore, of the 37 genes displayed in Table 6.23, 34 appear in the subset of 96 genes selected by the neural network method of Khan *et al.* (2001).

Note that the gene with index 1 194 is the only gene that is displayed in the left panel of Figure 6.14 but not included in Table 6.23. This gene has image clone identifier 859 359 and is the gene expression for Quinone oxidoreductase homolog (PIG3). Further investigation was performed to determine how many times the genes that were selected in every split of the NSC VS procedure appear in the correlation thresholding selection. This investigation indicated that using correlation thresholding, the gene with index 2 046 is selected at least 70% of the time, gene 1 003 is selected at least 65% of the time and gene 1 955 is selected at least 55% of the time.

Figure 6.15 below shows the shrunken differences d'_{jg} for the 37 significant genes from the SRBCT data set (reported in Table 6.23). These 37 genes have at least one non-zero difference in 80% of the 200 random splits of the SRBCT data set. The four different coloured bars in Figure 6.15 are the d'_{jg} for the four population groups in the SRBCT data set. These are obtained by soft-thresholding the 37 significant genes using NSC $\Delta = 3.3309$. Figure 6.15 is similar to Figure 3 plotted in the paper by Tibshirani *et al.* (2001).

Image clone ID 296 448 (index 187) has the maximum non-zero d'_{jg} of all the 37 genes in Figure 6.15, with a $d'_{jg} = 2.1899$ for RMS.

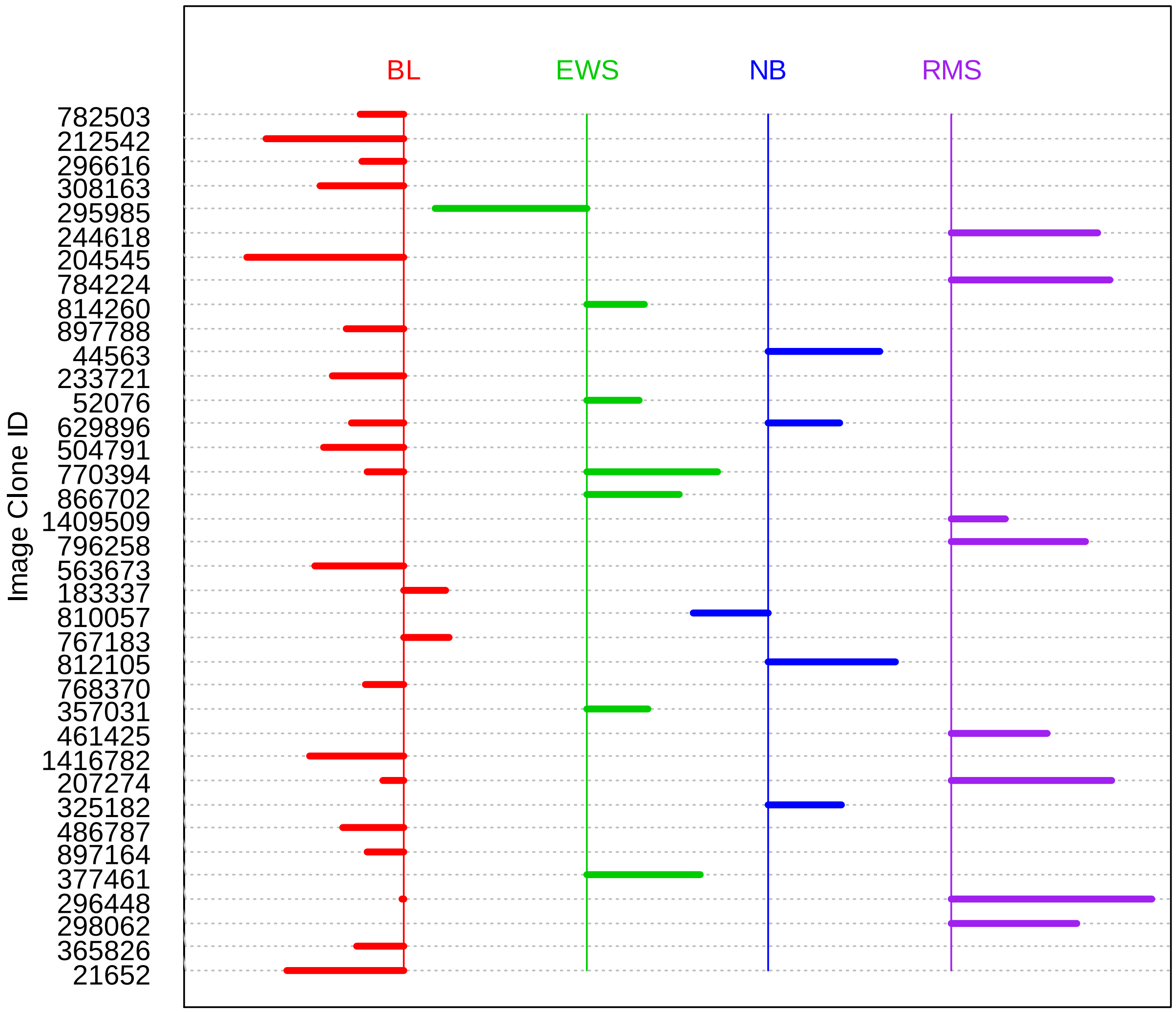

Figure 6.15: Shrunken differences d'_{jg} for 37 significant genes in the SRBCT data

Note that genes with shrunken differences d'_{jg} plotted to the left of the vertical population line, indicate that d'_{jg} is negative and therefore the gene is underexpressed in the population. On the other hand, genes with a positive shrunken difference d'_{jg} are plotted to the right of the vertical population line indicating that the gene is overexpressed in the population. The BL population has the largest number of genes (21 genes) with non-zero shrunken differences d'_{jg} of all the four population groups in Figure 6.15. Of these 21 genes, 2 have $d'_{jg} > 0$ and are overexpressed in BL and the remaining 19 genes are underexpressed in BL. Figure 6.15 illustrates that the gene with Image clone ID 810 057 (index 842) is underexpressed in NB and the gene with Image clone ID 295 985 (index 2 050) is underexpressed in EWS (which supports the results reported by Tibshirani *et al.*, 2001). The remaining six genes in EWS and four genes in NB plotted in Figure 6.15

have a positive d'_{jg} and are overexpressed. All eight genes with a non-zero d'_{jg} in RMS plotted in Figure 6.15 are positive and are overexpressed in RMS.

Figure 6.15 shows that the 37 significant genes are almost mutually exclusive in each class. There are only three significant genes that have non-zero shrunken differences d'_{jg} for more than one class. For example, the gene with Image Clone ID 629 896 (1 601) has a non-zero d'_{jg} for both the BL (red) and NB (blue) population groups (at values of -0.5552 and 0.7758 respectively). As mentioned previously this gene was not selected by NSC in the paper by Tibshirani *et al.* (2001).

In Figure 3 of Tibshirani *et al.* (2001), only EWS had a non-zero d'_{jg} for the gene with image clone ID 770 394, whilst in Figure 6.15 the BL (blue) and EWS (green) groups both have a non-zero d'_{jg} corresponding to the gene with image clone ID 770 394 (1 389). For BL the $d'_{jg} = -0.3866$ and the d'_{jg} is much larger for EWS at a value of 1.4364.

Finally, the gene with Image Clone ID 207 274 (509) has a non-zero d'_{jg} for both the BL (red) and RMS (purple) groups. In Figure 3 of Tibshirani *et al.* (2001), the gene with image clone ID 207 274 only has a non-zero d'_{jg} for RMS, however in Figure 6.15 the d'_{jg} for BL is -0.2134 which is very small compared to RMS $d'_{jg} = 1.7463$.

To conclude, from the results reported in Table 6.21 and 6.22 it is evident that for the SRBCT data set the NSC classifier applied to the 48 shrunken genes is the most accurate classification procedure with a test accuracy rate of 99.97%. Note that the NSC classifier developed in the paper by Tibshirani *et al.* (2001) achieved a perfect classification rate (100% accuracy) using 43 shrunken genes.

Papers by Khan *et al.* (2001) and Liu *et al.* (2009) make use of different base classifiers not investigated in this study which achieved 100% prediction accuracy on the SRBCT data set. The reader is referred to these paper for more details regarding these classification procedures.

6.3 COMPARING CLASSIFICATION PERFORMANCE ACROSS DATA SETS

This chapter considers several different classification procedures evaluated on six different high-dimensional microarray data sets. Hence, it is highly unlikely that there will be a single classification procedure that is consistently superior to the others on all data sets. Therefore, in order to determine which methods are better than others, and should be recommended for future applications, a rank was assigned to each classification method for each data set separately. This ranking was based on average test error rate over the 200 random splits of the data set. The ranking considers the different values of K in the KNN classifier and the C parameter in the SVM classifier as different procedures, *i.e.* cor-SVM with $C = 1$ could be ranked first while cor-SVM with $C = 1\ 000$ is ranked last. Although the number of features used plays an important role in evaluating the performance of a classification procedure, it was only taken into consideration in the rankings process in scenarios where two different procedures achieved the same test error rate. In such scenarios, the procedure with a smaller number of features used was ranked first. In the cases where more than one KNN procedure had the same test error rate and number of features used for different values of K, the procedure using the smaller value of K was considered superior. Similarly, in scenarios where SVM classification procedures yielded the same test error rate and used the same number of features for different values of C, the procedure using a smaller C parameter value was considered superior.

The following tables summarise the five top ranking procedures on the colon, leukemia, Sim1 and Sim3 data sets in terms of the average test error rate over the 200 random splits. Note that in total 55 procedures were applied to these four data sets.

Table 6.24: Top five classification procedures on the colon data set

Rank	Method	No. of features	Test Error	Standard Error	FNR
1	ttest-SVM ($C = 1$)	34	0.1612	0.0068	0.0715
2	cor-SPCA-KNN ($K = 5$)	16	0.1646	0.0069	0.0742
3	cor-KNN ($K = 3$)	706	0.1646	0.0069	0.0677
4	cor-SPCA-KNN ($K = 3$)	16	0.1669	0.0070	0.0727
5	cor-KNN ($K = 5$)	706	0.1669	0.0067	0.0662

Table 6.24 shows that the SVM classifier with $C = 1$ applied to the subset of 34 genes from the two-sample t-test gives the most accurate classification of the colon cancer data. It is interesting that the top procedure on the colon data set implements the SVM classifier, while four of the five top ranking procedures apply the KNN classifier with $K = 3$ or 5. Looking at the KNN classifier applied to the colon data set, the KNN classifier performs best when applied to the subsets of 16 SPCs using correlation thresholding. In Table

6.24, cor-SPCA-KNN ($K = 5$) and cor-KNN ($K = 3$) both have the same test error rate and the same standard error, but cor-SPCA-KNN is ranked in 2nd place as it used 97.73% less features than cor-KNN. Although the FNR was not considered in the ranking process, the last column of Table 6.24 reports that cor-KNN ($K = 5$) has the minimum FNR, and that furthermore cor-SPCA-KNN ($K = 5$) has a 9.66% larger FNR than cor-KNN ($K = 3$).

The top five classification procedures in the leukemia data set are reported in Table 6.25.

Table 6.25: Top five classification procedures on the leukemia data set

Rank	Method	No. of features	Test Error	Standard Error
1	SVM ($C = 1\ 000$)	3 571	0.0107	0.0017
2	SVM ($C = 10\ 000$)	3 571	0.0107	0.0017
3	cor-SVM ($C = 1$)	2 643	0.0110	0.0018
4	cor-SVM ($C = 1\ 000$)	2 643	0.0110	0.0018
5	cor-SVM ($C = 10\ 000$)	2 643	0.0110	0.0018

Table 6.25 clearly indicates that for the leukemia data set the SVM classifier using $C \in \{1, 1\ 000, 10\ 000\}$ is superior to KNN classification with $K \in \{1, 3, 5\}$. The SVM with $C = 1\ 000$ fitted to the full leukemia data is the most accurate and stable procedure. In Table 6.25, cor-SVM with $C \in \{1, 1\ 000, 10\ 000\}$ ranked 3rd ,4th and 5th, all achieving a test error rate of 0.0110. Although it is not shown in Table 6.25, the SVM with $C = 1$ fitted to all the genes in the leukemia data set achieved the same test error of 0.0110 and the same standard error as cor-SVM, but is ranked in 6th position as it used more features.

Unlike in the colon data set the comparison of the classification procedures in Table 6.25 shows that the SVM classifier yields a 3.12% higher test error when applied to a reduced set of correlation thresholding genes.

The top five procedures for Sim1 are summarised in Table 6.26 below. Four of the five classification procedures shown in Table 6.26 used the SVM classifier (two with $C = 1$) and the remaining procedure applied the DLDA classifier. Table 6.26 confirms that PCA-SVM with $C = 1$ has the smallest misclassification rate over the 200 random splits of the Sim1 data set. It is interesting that for Sim1, PCA is superior to NSC (which is location based), since Sim1 has a location difference between the two populations.

Table 6.26: Top five classification procedures on the Sim1 data set

Rank	Method	No. of features	Test Error	Standard Error
1	PCA-SVM ($C = 1$)	27	0.1594	0.0087
2	NSC-DLDA	128	0.1619	0.0083
3	NSC-SVM ($C = 1$)	128	0.1738	0.0084
4	NSC-SVM ($C = 1\ 000$)	128	0.1738	0.0085
5	NSC-SVM ($C = 10\ 000$)	128	0.1738	0.0084

The results in Table 6.26 suggest that the SVM classifier is superior to the KNN classifier on the Sim1 data set. The top ranking KNN procedure is cor-SPCA-KNN with $K = 5$, which is ranked 20th out of the 55 procedures applied to Sim1 with an average test error rate of 0.2094.

The ranking of the procedures for the Sim2 data set is temporarily excluded from the discussion and is reported later as not all 55 procedures are applied to Sim2.

The ranking of the 55 classification procedures on Sim3 are given in the table below.

Table 6.27: Top five classification procedures on the Sim3 data set

Rank	Method	No. of features	Test Error	Standard Error
1	PCA-SVM ($C = 1$)	27	0.1756	0.0084
2	PCA-SVM ($C = 1\ 000$)	27	0.1775	0.0087
3	PCA-SVM ($C = 10\ 000$)	27	0.1781	0.0087
4	SVM ($C = 1$)	1 000	0.1919	0.0080
5	DLDA	1 000	0.1950	0.0085

In Table 6.27, the top four ranked classification procedures on Sim3 applied the SVM classifier and the fifth applied the DLDA classifier. As in Table 6.26, the results in Table 6.27 suggest that the SVM classifier is superior to the KNN classifier on the Sim3 data set. Further analysis shows that the top ranking KNN procedure is cor-KNN with $K = 5$, which is ranked in 14th place (out of 55 procedures) with an average test error rate of 0.2256.

In general it appears that the top five ranking classification procedures are more accurate on Sim1 than on Sim3. From the results reported in Table 6.26 and Table 6.27, it can be concluded that for synthetic data sets (*viz.* Sim1 and Sim3) performing SVM classification using $C = 1$ on the 27 principal components gives the most accurate classification. However, PCA-SVM does not appear in the top five procedures on the real colon and leukemia data sets.

In order to determine a single best procedure on the binary data sets, the procedures are ranked over the different data sets according to each procedure's average test error rate over the 200 repetitions. The rankings range from 1 to 55, where 1 represents the most accurate procedure and 55 is the worst procedure. Only the binary data sets on which all 55 procedures were implemented, are included in the ranking and hence Sim2 is excluded. The 55 procedures considered in the thesis are ranked by computing the average of the test error rankings for the four binary data sets (*viz.* colon, leukemia, Sim1 and Sim3 data sets). The ranking of the procedures on Sim2 is given later in this section.

In scenarios where KNN procedures yielded the same test error rate for different values of K, the procedure using the smaller value of K was ranked higher. Similarly, in scenarios where SVM procedures yielded the same test error rate for different values of C, the procedure using a smaller C parameter value was ranked higher. Although the number of features used plays an important role in determining the performance of classification procedures, it was not included in the test error ranking. The rankings of all 55 classification procedures can be found in Appendix A.

Table 6.28: Top ten classification procedures on the binary data sets

		Test Error rankings for				
Rank	**Method**	**Colon**	**Leukemia**	**Sim1**	**Sim3**	**All four**
1	SVM (C =1 000)	7	1	17	9	8.50
2	cor-SVM ($C = 1$)	10	3	13	11	9.25
3	SVM ($C = 10\ 000$)	9	2	16	10	9.25
4	cor-SVM ($C = 1\ 000$)	8	4	14	12	9.50
5	cor-SVM ($C = 10\ 000$)	11	5	15	13	11.00
6	ttest-SVM ($C = 1$)	1	12	7	33	13.25
7	cor-SPCA-KNN ($K = 5$)	2	15	20	17	13.50
8	NSC-DLDA	12	25	2	15	13.50
9	SVM ($C = 1$)	41	6	10	4	15.25
10	cor-KNN ($K = 5$)	5	17	26	14	15.50

Table 6.28 indicates that the most accurate classification procedure out of the 55 considered in the thesis is the SVM classifier with $C = 1\,000$ applied to all genes in the data sets. In Table 6.28 it appears that the accuracy of the SVM with $C = 1\,000$ and 10 000 starts to degrade as the number of features used is reduced from the original p in the data set. The results reported in Table 6.28 contradicts the conclusion that variable selection is very much worthwhile for SVMs. Though, it should be kept in mind that this conclusion is based on the analysis of only a few data sets with a few variable selection methods. However, when applying the SVM to a reduced subset of genes, it appears that the classifier is superior when applied to the reduced set of genes using correlation thresholding rather than the two-sample t-test (cor-SVM is ranked 2nd while ttest-SVM is ranked 6th with $C = 1$).

In summary, Table 6.28 confirms that in general the SVM classifier outperforms the KNN classifier for the $p \gg N$ microarray binary data sets considered in this thesis. The SVM classifier is applied in seven of the top ten classification procedures, while the KNN classifier only appears once in cor-SPCA-KNN (with $K = 5$) which is ranked in 7th place in Table 6.28.

The results reported in Table 6.28 indicate that overall correlation thresholding is the most accurate variable selection approach (and superior to all other dimension reduction techniques that were investigated) for the SVM classifier applied to the binary data sets. The results for the colon data set contradict this conclusion as the two-sample t-test (ranked 1st) is superior to correlation thresholding (ranked 10th) for the SVM classifier with $C = 1$. This may be explained by looking at the optimal thresholding values selected in Section 5.6 for the SVM classifier with $C = 1$ on the colon data set. In Table 5.2, the selection frequencies are approximately evenly distributed over the three largest correlation candidate values (0.3026, 0.4039 and 0.5053). The correlation threshold values used in the SVM classifier with $C = 1$ was 0.3026, which was selected in 31% of the 100 random splits. The results for the t-test reported in Table 5.8 clearly show that for the SVM classifier with $C = 1$ the optimal p-value $= 0.001$ as it was selected in 67% of the 100 random splits of the colon data set.

On average for the four binary data sets, SPCA with correlation thresholding yields the most accurate subset of genes for the KNN classifier with $K \in \{1, 3, 5\}$, placed in seventh position in Table 6.28. The results further indicate that correlation thresholding alone yields the second most accurate subset of genes for the KNN classifier. However, results for the KNN classifiers applied to the leukemia data set (given in Table 6.9) contradict this conclusion as the ttest-SPCA-KNN procedure with 34 SPCs outperformed the cor-SPCA-KNN procedure with 41 SPCs. The optimal correlation threshold value selected from Table 5.4 is very small (0.1000 for $K = 1$ and 3 and 0.2469 for $K = 5$). Therefore, the performance of the KNN classifier in cor-SPCA-KNN was negatively affected by the small reduction in irrelevant genes in the leukemia data set (on average 2 643 genes were selected at the correlation threshold value of 0.100 and 1 345 for 0.2469). Table 5.9 shows that the optimal p-value $= 0.001$ in the two-sample t-test for $K \in \{1, 3, 5\}$ in the KNN classifier, which is very strict and resulted in only 538 genes remaining in the model. Therefore, in the leukemia data set it is no surprise that KNN on the ttest-SPCs outperformed KNN on cor-SPCs. Therefore, from the results one can conclude that in general the correlation thresholding yields a more accurate subset of genes than the two-sample t-test in the binary setting.

DLDA applied to the reduced set of NSC shrunken genes is ranked 8th over the four binary data sets. In general, the NSC-DLDA procedure is superior to DLDA on all the genes. However, DLDA applied to all the genes in the Sim3 data set outperforms the DLDA classifier applied to the subset of NSC selected genes. The NSC procedure reduces the subset of genes by at least 87.20% and besides in Sim1 the reduced subset of genes is smaller than N (overcoming the curse-of-dimensionality). Furthermore, the DLDA classifier is easy to implement but according to Dudoit *et al.* (2002:85) it ignores the correlations amongst the expression levels for different genes which may prove to be problematic. Consider now the performance of the LDA, DLDA and NSC classifiers on the subset of genes selected by the NSC VS procedures on the binary data sets (excluding Sim2). For all four data sets, the DLDA classifier was the best performing method followed by the NSC classifier. The LDA classifier (NSC-LDA) was the least accurate procedure.

The following process was conducted in order to incorporate the number of features used in the rankings assigned to the four binary data sets. The number of features used in each of the top ten classification procedures (in Table 6.28) was recorded for each data set. Then each procedure was given a ranking from 1 to 10 based on the number of features used, where the procedure that made use of the smallest number of features is ranked first. In the scenario where two different classification procedure both used the same number of features, the test error rate was taken into consideration. Finally, when more than one procedure in a given data set used the same number of features and achieved the same error rate, the procedure with the smallest value of K or C was considered superior. The feature ranking was then averaged over the four data sets.

The feature ranking procedure was not applied to all 55 classification procedures to eliminate the chance that a procedure that used very few features but has a high test error rate is recorded as the top ranking procedure. For example, fastKNN with $K = 2$ makes use of only two features (which is the minimum) but achieved a very large test error rate for all four data sets and thus ranked last in terms of test error.

The ranking according to test error rate and the ranking according to the number of features used for each of the ten classification methods (in Table 6.28) are depicted in Figure 6.16.

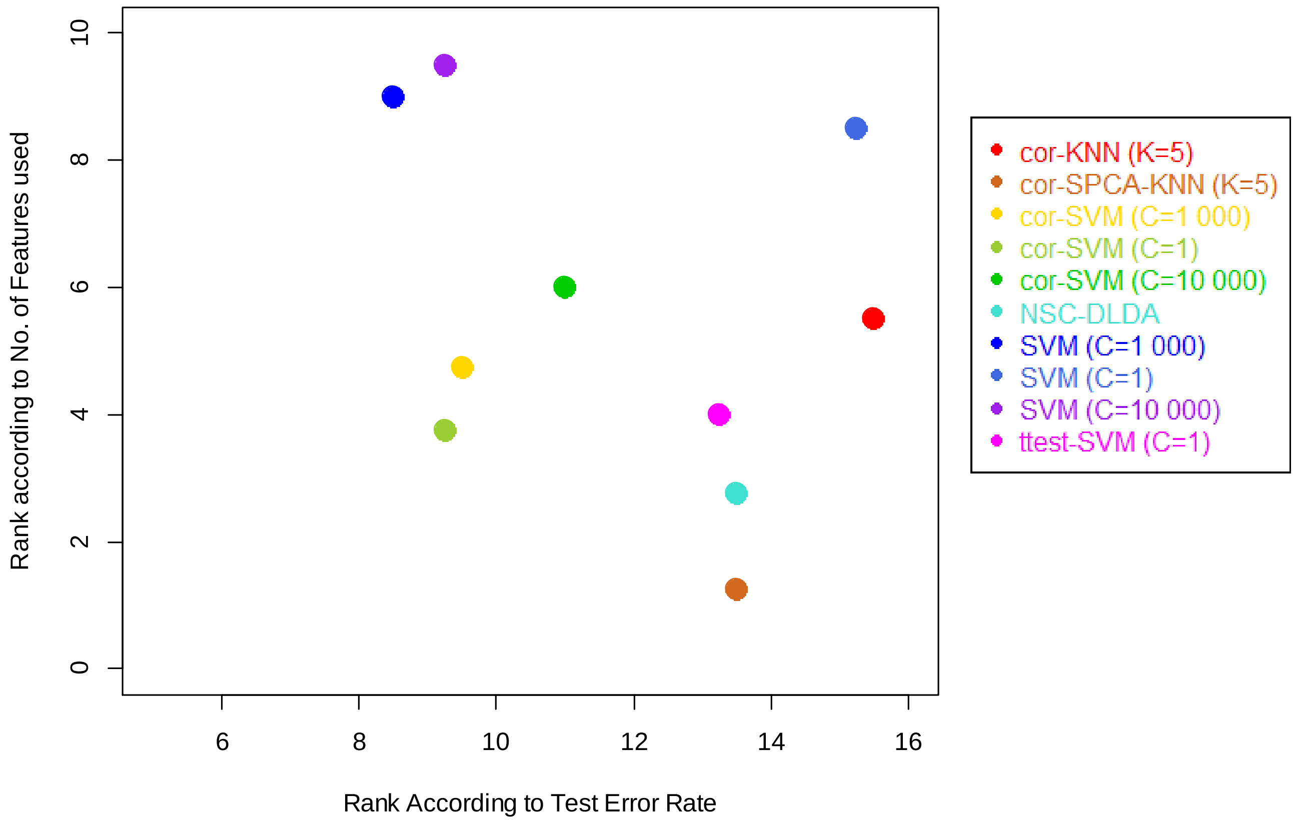

Figure 6.16: Ranking of the top ten classification procedures

The ideal classification procedure is plotted in the bottom left corner, indicating a low ranking according to both number of features used and average test error rate. Therefore, the best classification procedure in Figure 6.16 is cor-SVM with $C = 1$ as it is the closest procedure to the bottom left corner. It is clear in Figure 6.16 that SVM with $C = 1\,000$ fitted to all the genes in the four data sets is the most accurate procedure; however, it was heavily penalised as it used all of the genes in the data set, and has the second largest feature ranking. On the other hand, cor-SPCA-KNN with $K = 5$ is ranked first in terms of the number of features (making use of the smallest number of features of the procedures in Figure 6.16), however its overall ranking is penalised by the procedures high test error rate.

Ranking of the procedures for the Sim2 data set is reported separately in Table 6.29. This is because only 19 classification procedures were implemented on this data set instead of all 55 considered for the other binary data sets.

Table 6.29: Top five classification procedures on the Sim2 data set

Rank	Method	No. of features	Test Error	Standard Error
1	SVM (C =1)	1 000	0.3994	0.0092
2	DLDA	1 000	0.4275	0.0108
3	fastKNN ($K = 5$)	10	0.4313	0.0112
4	SVM (C =1 000)	1 000	0.4331	0.0110
5	SVM (C =10 000)	1 000	0.4331	0.0110

The rankings in Table 6.29 support the rankings for Sim1 and Sim3, indicating that the SVM and DLDA classifiers are more accurate than the KNN classifier when applied to the synthetic high-dimensional data sets.

This is the first time that the fastKNN procedure appears in the top five classification procedures. Whilst the results in Table 6.29 contradict the general conclusion that the ordinary KNN classifier outperforms the fastKNN classifier, the fastKNN procedure with $K = 5$ (ranked third in Table 6.29) still achieved a very high error rate of 0.4313. Furthermore, the KNN classifier in Sim2 was applied to all 1 000 genes in the data set even though it was known to only have 100 relevant genes, and hence the poor performance of KNN is expected. The highest ranking KNN classification procedure on Sim2 is the 3-NN classifier applied to the full Sim2 data set which appears in 8th position.

From the results summarised in Table 6.24 through 6.29 it can be concluded that for binary microarray data the KNN classifier is more accurate for $K = 3$ or 5 than for $K = 1$.

Comparing the results of the 5-NN, SVM with $C = 1$ 000 and the DLDA classifiers applied to all the genes in the six binary microarray data sets leads to the following conclusions. Relatively large improvements in classification accuracy were observed if the SVM classifier with $C = 1\,000$ rather than the 5-NN classifier was applied to all the original binary data sets. Clearly, the SVM classifier is preferable to the 5-NN classifier in the microarray data sets. The rankings for the full four binary data sets considered in Table 6.28 (included in Appendix A) indicate that on average the SVM with $C = 1\,000$ outperforms DLDA, both of which are superior to the 5-NN classifier (this also holds true for the leukemia data set). However, in the colon data set the 5-NN classifier applied to all the genes outperformed DLDA. Furthermore, in the Sim1 and Sim3 data sets the DLDA classifier is more accurate than SVM applied to all the genes. Since the two groups in Sim1 and Sim3 were generated using a normal distribution it is not surprising that DLDA performs well as it assumes a normal distribution of values for each class.

This section now ranks the performance of the 22 multi-class classification procedures considered on the SRBCT data set. The rankings of all 22 classification procedures applied to the SRBCT data set is provided in Appendix A.

Table 6.30: Top five classification procedures on the SRBCT data set

Rank	Method	No. of features	Test Error	Standard Error
1	NSC	48	0.0003	0.0003
2	NSC-KNN ($K = 1$)	48	0.0006	0.0004
3	NSC-KNN ($K = 5$)	48	0.0008	0.0005
4	NSC-KNN ($K = 3$)	48	0.0025	0.0009
5	KNN ($K = 3$)	2 308	0.0933	0.0045

It is evident from the results reported in Table 6.30 that the NSC procedure selects the most accurate subset of genes on the SRBCT data set for both the NSC and KNN classifiers. The results given Table 6.30 show that the NSC classifier is superior to the KNN classifier for $K \in \{1, 3, 5\}$ applied to the subset of 48 NSC selected genes. In Table 6.30, NSC-KNN with $K = 1$ is the best performing KNN classification procedure on the SBRCT data set. However, on average the 3-NN classification procedures have the minimum average test error rate amongst the seven KNN classifiers in Table 6.21, followed by the 5-NN classifier and finally the 1-NN classifier.

The results of the study on the SRBCT data set were encouraging: the KNN classifier with $K \in \{1, 3, 5\}$ is more accurate on the reduced subset of 43 genes than on all 2 308 genes, supporting the conclusion that variable selection is important when applying the KNN classifier to high-dimensional data sets.

In this chapter, smaller average test error rates were generally observed for the 5-NN classification procedures compared to the 3-NN and 1-NN procedures. The 3-NN classification procedures only achieve a smaller average test error rate than the 5-NN classifier for the Sim2 and SRBCT data sets, but the difference is not significant (0.90% and 1.01% respectively). Therefore it can be concluded that the KNN classification procedure with $K = 5$ is preferable to the KNN classifier with $K = 1$ and 3 on the microarray data sets. It is important to note the detrimental effect of irrelevant variables on the accuracy of the KNN classifier. This can be seen as the most accurate KNN classification procedure applied to the binary data sets considered in the thesis is cor-SPCA-KNN (with $K = 5$) which is ranked in 7th place in Table 6.28. Furthermore, it is clear from the results that in all cases the test error rates markedly increased by the inclusion of irrelevant variables in the KNN classifier. Although the KNN classifier is only in positions 7 and 10 in Table 6.28, it is simple to apply, intuitive and can handle interactions between the genes in a microarray data set albeit in a “black-box” manner which provides very little insight into the data set (Dudoit *et al.,* 2002:85).

Regarding the performance of the PCA and SPCA dimension reduction techniques presented in this chapter, it is difficult to express a general preference for either. SPCA as explained in Section 4.3 aims to denoise the response, resulting in a dramatic dimension reduction while still retaining most of the variation in the data set. Therefore, using the correct measure to rank the p predictor variables in Step 1 of SPCA is vital to the success of SPCA.

From the results for the KNN classification procedures in Table 6.12 (for Sim1) and Table 6.18 (for Sim3), it can be seen that the SPCA dimension reduction technique using both correlation and t-test thresholding outperformed ordinary PCA for all $K \in \{1, 3, 5\}$ in the KNN classifier. However, a relatively large improvement in classification accuracy were observed for PCA rather than SPCA (both cor-SPCA and ttest-SPCA) for all values of the cost parameter in SVM classifier on the Sim1 and Sim3 data sets, given in Table 6.13 and 6.19 respectively. Due to the practical limitations previously mentioned, SPCA was not implemented on the Sim2 data set and thus a comparison of PCA versus SPCA as a dimension reduction technique could not be discussed.

An interesting pattern emerged in the results for the KNN and SVM classifiers on the colon data set presented in Section 6.2.1. In the scenario where the KNN or SVM classifier performed better on the subset of genes from correlation thresholding than on all the genes, the same KNN or SVM classifier also performed better on the subset of genes from cor-SPCA than PCA. This highlights the importance of implementing the correct measure (and size of the measure) in Step 1 of SPCA and its influence on the performance of the SPCA procedure. For example, consider Table 6.3 for $K = 5$, the cor-KNN procedure outperformed the KNN applied to all the genes and the cor-SPCA-KNN procedure outperformed PCA-KNN. From Section 6.2.1 it can been seen that for the colon data set the PCA dimension reduction approach is outperformed by ttest-SPCA procedure only when the 3-NN and 5-NN classifiers are applied.

Turning attention to the results for the KNN and SVM classification procedures shown in Table 6.9 and 6.10, it appears that PCA is not an appropriate dimension reduction technique on the leukemia data set as it leads to larger error rates for both the KNN and SVM classifiers in comparison to the other procedures considered. It is interesting to see that the cor-SPCA procedure was superior to PCA for all value of K in the KNN classifier and C in the SVM. Futhermore, besides in the cases where $C = 1\ 000$ and 10 000 in the SVM classifier the ttest-SPCA procedure also outperformed PCA. Remember that the SVM classifier with $C = 1\ 000$ and 10 000 fitted to all the genes in the leukemia data set were tied as the most accurate procedures.

The high optimal correlation threshold value of 0.7253 selected by the KNN classifier in the SRBCT data set resulted in a subset comprised of too few variables, only seven on average. The fact that the KNN fit to all the genes in the SRBCT data set outperforms the cor-KNN procedure (given in Table 6.21) suggests that correlation thresholding is too strict on the SRBCT data set resulting in too few variables to provide sufficient information for accurate classification. Hence, it is not surprising that PCA-KNN outperforms SPCA with correlation thresholding on the SRBCT data set. Finally it should be noted that in Table 6.21, PCA-KNN also outperforms cor-KNN which further supports the conclusion.

The leukemia and SRBCT data sets did not pose very difficult classification problems and achieved relative low test error rates. The average test error rates for the top five classification procedures in Table 6.25 for leukemia was 0.0109, while the average test error rate for the SRBCT classification procedures in Table 6.30 is 0.0195. The colon cancer, Sim1 and Sim3 data sets proved to be more challenging and the top five classification procedures presented in Table 6.24, 6.26 and 6.27 had a combined average test error rate of 0.1648, 0.1685 and 0.1836 respectively. The classification procedures considered in the thesis appear to be unsuitable for application to the Sim2 data set, as the top five classification procedures for the Sim2 data set presented in Table 6.29 had a combined average test error rate of 0.4249.

An interesting question to consider concerns the classification error rates that can be achieved on the simulated data sets by applying the classification procedures under discussion only to the genes known to be relevant. Naturally these error rates will be appreciably lower than the error rates achievable by any procedure acting without knowledge of which genes are relevant. Also, in any practical application no procedure will have such information available. The difference between the error rates obtained by applying a classifier only to the relevant genes and the same classifier applied to the genes identified by a variable selection method will provide an indication of the success with which the variable selection has been performed.

Consider therefore the three simulated data sets. Each of these data sets, Sim1, Sim2 and Sim3, contains 100 known relevant variables and an additional 900 noise variables.

Table 6.31 below provides the results of the KNN classification method for $K \in \{1, 3, 5\}$ applied only to the 100 relevant genes in the Sim1 data set.

Table 6.31: Results of the KNN classification procedures on Sim1

Method	$K = 1$			$K = 3$			$K = 5$		
	No. of feature	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**
KNN	100	0.1263	0.0079	100	0.0525	0.0051	100	0.0481	0.0050
fastKNN	2	0.1550	0.0078	6	0.0856	0.0064	10	0.0506	0.0052
Average		0.1406	0.0079		0.0691	0.0057		0.0494	0.0051

The 5-NN classifier applied to all 100 relevant genes in Sim1 is the most accurate and stable procedure in Table 6.31. The test error rate of the KNN classifier decreased by 67.37%, 84.95% and 83.65% respectively for $K \in \{1, 3, 5\}$ when implemented on only the relevant genes in Sim1 as opposed to the full data set. The fastKNN procedure's test error rate decreased by over 70% for all $K \in \{1, 3, 5\}$ when applied to only the relevant genes in Sim1. It is clear that the quality of a KNN classifier improves dramatically when all the noise variables in the data set can be eliminated.

The results summarised in Table 6.31 suggest that for the Sim1 data set none of the VS or dimension reduction approaches underlying the results in Table 6.12 are successful in determining the relevant genes.

The results for the SVM and fastKNN-SVM classification procedures for $C \in \{1, 1\,000, 10\,000\}$ applied to only the 100 relevant genes in the Sim1 data set are summarised in Table 6.32.

Table 6.32: Results of the SVM classification procedures on Sim1

Method	**No. of features**	$C = 1$		$C = 1\,000$		$C = 10\,000$	
		Test Error	**Std. Error**	**Test Error**	**Std. Error**	**Test Error**	**Std. Error**
SVM	100	0.0213	0.0036	0.0194	0.0034	0.0188	0.0034
fastKNN-SVM	10	0.0606	0.0056	0.0975	0.0071	0.0969	0.0071
Average		0.0409	0.0046	0.0584	0.0053	0.0578	0.0053

As with the full Sim1 data set, the results in Table 6.32 indicate that the SVM classifier is superior to the KNN classifier. The SVM classifier with $C = 10\,000$ is the best classification procedure in Table 6.32.

Overall, there is a significant improvement in the accuracy of both the SVM and fastKNN-SVM procedure for all for $C \in \{1, 1\,000, 10\,000\}$ when applied to only the 100 relevant genes in the Sim1 data set. The test error rate of the SVM classifier decreased by 88.67%, 90.06% and 90.35% respectively for $C \in \{1, 1\,000, 10\,000\}$ when implemented on only the relevant genes in Sim1 as opposed to the full data set. Additionally, the standard errors for the SVMs for all $C \in \{1, 1\,000, 10\,000\}$ decreased by over 50% when applied to only the relevant genes instead of all 1 000 genes. The fastKNN-SVM procedure (with $K = 5$) achieved a slightly lower decrease in test error rate of over 78% for all $C \in \{1, 1\,000, 10\,000\}$ when applied to only the relevant genes in Sim1.

Table 6.33 provides the result of DLDA on the 100 relevant genes in the Sim1 data set.

Table 6.33: Results of the DLDA classifier on Sim1

Method	**No. of features used**	**Test Error Rate**	**Standard Error**
DLDA	100	0.0163	0.0034

The DLDA classifier applied to the 100 relevant genes is 90.81% more accurate than when applied to all 1 000 genes in Sim1.

In summary, the best classification procedure on the Sim1 data set with all the 1 000 genes is PCA-SVM (with $C = 1$) achieving a test error rate of 0.1594. On the other hand, the best classification procedure considering only the 100 known relevant genes in Sim1 is the DLDA classifier which achieves a test error rate of 0.0163. It seems from these results that the VS and dimension reduction approaches studied in this thesis are not very successful in selecting the relevant variables in the Sim1 data set. In addition, it is clear that the classifiers are all quite sensitive to the presence of noise variables.

Table 6.34 below provides the results of the KNN classification method for $K \in \{1, 3, 5\}$ applied only to the 100 relevant genes in the Sim2 data set.

Table 6.34: Results of the KNN classification procedures on Sim2

Method	$K = 1$			$K = 3$			$K = 5$		
	No. of feature	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**	**No. of feature**	**Test Error**	**Std. Error**
KNN	100	0.3063	0.0103	100	0.4231	0.0119	100	0.4113	0.0111
fastKNN	2	0.4688	0.0127	6	0.4394	0.0121	10	0.4250	0.0118
Average		0.3875	0.0115		0.4313	0.0120		0.4181	0.0114

The 1-NN classifier applied to the 100 relevant genes in the Sim2 data set yields the smallest test error rate of 0.3036, and is also the most stable procedure in Table 6.34. When comparing the results in Table.6.34 to those in Table 6.15 for the full Sim2 data set, it is interesting that the 1-NN classifier's test error rate decreases by approximately 44.32% when the irrelevant genes are removed from Sim2, while the 3-NN and 5-NN test error rates only decrease by 8.51% and 16.07% respectively.

The accuracy of the fastKNN procedure improves by a smaller amount (2.98%, 1.40% and 1.45% for $K \in \{1, 3, 5\}$ respectively) when applied to the subset of relevant genes in Sim2 as opposed to the full Sim2 data set reported in Table 6.15.

However, one should note that the improvement in the KNN classifier applied to only the relevant genes in Sim2 (and not the full set) is substantially smaller than the results for Sim1.

The results for the SVM and fastKNN-SVM classification procedures for $C \in \{1, 1\,000, 10\,000\}$ applied to only the 100 relevant genes in the Sim2 data set are summarised in Table 6.35.

Table 6.35: Results of the SVM classification procedures on Sim2

Method	**No. of features**	**$C = 1$**		**$C = 1\,000$**		**$C = 10\,000$**	
		Test Error	**Std. Error**	**Test Error**	**Std. Error**	**Test Error**	**Std. Error**
SVM	100	0.4175	0.0091	0.4113	0.0117	0.4106	0.0118
fastKNN-SVM	10	0.4025	0.0106	0.4456	0.0128	0.4438	0.0121
Average		0.4100	0.0099	0.4284	0.0123	0.4272	0.0120

The results summarised in Table 6.35 show that the best SVM procedure is the SVM with $C = 1$ applied to the 10 features extracted from only the relevant genes using fastKNN with $K = 5$. It is interesting that when applied to the full Sim2 data set, the accuracy of the SVM increases as the value of C increases. On the other hand, when considering only the 100 relevant genes in Sim2, the accuracy of SVM decreases as the value of C increases.

Finally, comparing the results from Table 6.35 to that of Table 6.16 it is clear that for $C = 1$ the SVM classifier is 4.43% more accurate on the full Sim2 data set than on the 100 relevant genes in the Sim2 data set. On the other hand, for $C = 1\,000$ and 10 000 the SVM classifier is more accurate (5.05% and 5.19% respectively) on the 100 relevant genes than the full Sim2 data set.

Table 6.36 provides the results DLDA on the 100 relevant genes in the Sim2 data set.

Table 6.36: Results of the DLDA classifier on Sim2

Method	**No. of features used**	**Test Error Rate**	**Standard Error**
DLDA	100	0.4269	0.0105

It is interesting that the test error rate of the DLDA classifier only decreases by 0.15% when it is applied to the 100 relevant genes as opposed to the full Sim2 data set.

The best classification procedure on the Sim2 data set with all 1 000 genes is the SVM with $C = 1$ fit to all the genes achieving a test error rate of 0.3394. However, the 1-NN classifier is the best procedure when considering only the 100 relevant genes in the Sim2 data set, achieving a test error rate of 0.3036.

Table 6.37 below provides the results of the KNN classification method for $K \in \{1, 3, 5\}$ applied only to the 100 relevant genes in the Sim3 data set.

Table 6.37: Results of the KNN classification procedures on Sim3

Method	$K = 1$			$K = 3$			$K = 5$		
	No. of feature	Test Error	Std. Error	No. of feature	Test Error	Std. Error	No. of feature	Test Error	Std. Error
KNN	100	0.2338	0.0094	100	0.1919	0.0085	100	0.1831	0.0086
fastKNN	2	0.3050	0.0098	6	0.2600	0.0096	10	0.2313	0.0092
Average		0.2694	0.0096		0.2259	0.0090		0.2072	0.0089

The 5-NN classifier applied to the 100 relevant genes in the Sim3 data set achieves the smallest test error rate of 0.1831 in Table 6.37. The 5-NN classifier applied to the relevant genes in Sim3, yields a 26.57% smaller test error rate than when applied to the full Sim3 data set. Overall when comparing the results in Table 6.37 to those in Table 6.18 it is evident that the accuracy of the KNN classifier for $K \in \{1, 3, 5\}$ improves when applied only to the relevant genes in the Sim3 data set.

The results for the SVM and fastKNN-SVM classification procedures for $C \in \{1, 1\,000, 10\,000\}$ applied to only the 100 relevant genes in the Sim3 data set are summarised in Table 6.38.

Table 6.38: Results of the SVM classification procedures on Sim3

Method	No. of features	$C = 1$		$C = 1\,000$		$C = 10\,000$	
		Test Error	Std. Error	Test Error	Std. Error	Test Error	Std. Error
SVM	100	0.1594	0.0088	0.1138	0.0076	0.1144	0.0076
fastKNN-SVM	10	0.2844	0.0092	0.2444	0.0100	0.2506	0.0103
Average		0.2219	0.0090	0.1791	0.0088	0.1825	0.0090

The results summarised in Table 6.38 show that the best procedure is the SVM classifier with $C = 10\,000$ applied to the 100 relevant genes in Sim3. Comparing the results obtained when applying the SVM classification methods to the full Sim3 data set (summarised in Table 6.16), it is clear from Table 6.38 that the accuracy of the SVM improves for all $C \in \{1, 1\,000, 10\,000\}$ when applied to only the relevant genes in Sim3.

Table 6.39 provides the results of the DLDA classifier on the 100 relevant genes in the Sim3 data set.

Table 6.39: Results of the DLDA classifier on Sim3

Method	No. of features used	Test Error Rate	Standard Error
DLDA	100	0.1300	0.0081

From the results in Table 6.39 it appears that the test error rate of DLDA decreases by 33.33% when applied to the 100 relevant genes in Sim3 as opposed to the full Sim3 data set.

As mentioned previously, PCA-SVM with $C = 1$ is the best classification procedure on the full Sim3 data set yielding a test error rate of 0.1756. From the results summarised in Table 6.37 through 6.39 it can be concluded that the RBF SVM classifier with $C = 10\ 000$ applied to the 100 relevant genes in Sim3 is the best procedure, yielding a test error rate that is 34.88% smaller than the best procedure on the full data set.

In summary, it can be concluded that the classifiers considered above greatly benefit when applied to the known relevant genes in the Sim1 and Sim3 data sets, as opposed to the full data sets. However, there is not a substantial difference in the performance of a classifier on the Sim2 data set when applied only to the relevant genes as opposed to the full Sim2 data set. For the SVM with $C = 1$, the classifier performs better when all the irrelevant variables are included in the data set.

6.4 SUMMARY AND CONCLUSION

This chapter involved a comprehensive study of the different classification procedures for high-dimensional microarray data sets, focusing on the test error rate to evaluate the performance of each procedure. However, there are other factors that contribute to the merits of the classification procedures, namely the simplicity of the classifier, the number of features used, and insight gained into the predictive structure of the data set (Dudoit *et al.,* 2002:85). In mircorarray data sets, gene interactions are important biologically as they may contribute to class distinctions; hence applying a classifier that ignores the interactions is not desirable.

It should be noted that leukemia and SRBCT were low error rate data sets and the different procedures achieved impressive results on these data sets. The colon cancer, Sim1 and Sim3 data sets were a bit more challenging, yielding higher average test error rates. Finally, the classification procedures performed dismally on the Sim2 data set.

Overall, the most accurate classification procedure considered in the thesis is the NSC classification procedure applied to the SRBCT data set, achieving only a single misclassification over the 200 random splits of the data set, which corresponds to a 99.97% classification accuracy. This should however be interpreted in light of the fact that the SRBCT data set is easy to classify.

The findings in this chapter indicate that the SVM classifier with $C = 1\,000$ applied to the full binary data sets ranked the best out of all the 55 classification procedures considered on the colon, leukemia, Sim1 and Sim3 binary data sets. The SVM classifier with $C = 1\,000$ is also the best classifier on the leukemia data set. From the results in Table 6.28 it appears that the accuracy of the SVM starts to degrade as the number of features used is reduced from the original p in the data set. This contradicts the conclusion that variable selection is worthwhile for SVMs. This may be attributed to the fact that the location based VS procedures considered in this thesis do not perform well with SVMs and it may be worthwhile to look at VS procedures that are known to perform well with SVMs such as the so-called recursive feature elimination. Since one of the main drawbacks of applying RBF SVMs is the lack of interpretability in the resulting models, it is important to investigate VS procedures that are known to perform well with SVMs. Besides being the most accurate classifier considered in the thesis, the SVM classifier can handle high-dimensional data sets without increasing the learning complexity. However, as mentioned in Section 2.5, compared to the other classifiers considered in the thesis the SVM classifier is difficult to apply, requiring the correct specification of several tuning parameters. It should be borne in mind that although the cost parameter value was set to $C \in \{1,1\,000,\ 10\,000\}$ in the analysis, the RBF kernel hyperparameter, γ, was fine-tuned in the SVM according to the optimal value determine on each split of the data sets. The values of γ used in the SVM classification were not reported for each split of the data sets. It would be interesting to investigate the change of performance of the RBF SVM classifier if a default value of $\gamma = \frac{1}{p}$ was used.

The accuracy of the KNN classifier on the six data sets was computed for three different values of $K \in \{1, 3, 5\}$. In this chapter, smaller average test error rates were generally observed for the 5-NN classification procedures, over the 3-NN and 1-NN classification procedures. It is important to note the detrimental effect of irrelevant variables on the accuracy of the KNN classifier. The results showed that when applied to all the genes in the binary data sets, in general the SVM classifier is superior to the KNN classifier in the microarray data sets which supports the statements by Hastie *et al.* (2009) that the SVM classifier can handle high-dimensional data sets without increasing the learning complexity.

The results of the empirical study suggest that in general the majority voting rule used in the ordinary KNN classifier is superior to the weighted voting rule used in the fastKNN classifier. Both fastKNN feature extraction and classification performed relatively poorly on the six data sets considered, reflected in relatively large test errors recorded across all the data sets. On the whole, the fastKNN feature extraction method presented in this thesis is the worst performing dimension reduction technique. The results for using fastKNN as a feature extraction method for $K = 5$ confirm what is intuitively expected as the fastKNN classifier is more accurate than the SVM classifier when applied to the features generated using fastKNN for most binary data sets. The results reported for all the data sets in this chapter summarise that the accuracy of the new features generated by fastKNN increased as the value of K increased. This suggests that the dismal performance of the fastKNN feature extraction method may be attributed to the generation of too few new features to provide sufficient information for accurate classification. This conclusion is strengthened by the improved performance of the fastKNN feature extraction method for the SRBCT data set for $K = 5$ as the procedure generates 20 new features (5×4) as opposed to only 10 as in the binary scenario. The effect of larger values of K ($K > 5$) on the accuracy of the fastKNN feature extraction method could be investigated in future research.

DLDA is the best performing linear based classifier considered in this thesis. The DLDA classifier performed well in the simulated data sets as its assumption of a normal distribution of values for each group was correct in these cases. In general, the performance of the DLDA classifier noticeably improved when applied to a reduced subset of genes from the NSC VS procedure.

The results in this chapter indicate that different variable selection and dimension reduction techniques perform better for the KNN and SVM classifiers on different data sets. Nevertheless, overall it can be concluded that in general the correlation thresholding yields a more accurate subset of genes than the two-sample t-test in the binary setting. On average for the four binary data sets, SPCA with correlation thresholding yields the most accurate subset of genes for the KNN classifier with $K \in \{1, 3, 5\}$. The results further indicate that correlation thresholding alone yields the second most accurate subset of genes for the KNN classifier.

In the SRBCT data set, as mentioned above, the NSC VS procedures yield the most accurate KNN classification. The impressive performance of both the NSC classification and selection procedure on the SRBCT data reported in this chapter may be attributed to the fact that the LOOCV frequencies for the NSC Δ threshold value using both the KNN and NSC classifiers were less dispersed over the range of 15 possible candidate values than in the case of the binary data sets. The performance of the NSC procedure was already hindering by the fact that the Δ threshold value used for each data set was the candidate value with the highest frequency in LOOCV in attempt to reduce computations. The poor performance may also be attributed to the selection of too few variables to provide sufficient information for accurate class separation. Similarly, forcing the NSC procedure to select too many irrelevant variables that are not applicable in a random split of the data set may also hinder the classification accuracy. Hence, the performance of the NSC VS procedure would likely improve if the Δ threshold value was determined once the data is split into a training and test case, and then re-determine the optimal thresholding value for each split.

The empirical study indicated the need to include dimension reduction techniques such as PCA and SPCA that identify linear combinations that maximise the variation in the data set, especially in the case of the non-parametric KNN classifier. The results confirm the benefits of using SPCA over ordinary PCA, however the measure used to rank the predictor variables in Step 1 of SPCA is crucial to ensuring the accuracy of the procedure. There is scope for further research into different measures in addition to the correlation coefficient and the p-value two-sample t-test that were used in this thesis. An example is the score statistic discussed in Section 3.3.3. Another possibility is to implement hard-thresholding in SPCA, which is briefly described in Section 4.3, instead of soft-thresholding which is normally used.

A fundamental problem with resampling methods occurs when a data set has a small number of samples. This is often the case in certain fields, such as for example in genomics. In small data settings, further splitting at each step results in high variance. The results reported in this chapter support this conclusion, as the average of the standard errors for the top five classification procedures reported for the data sets are ordered according to N as follows. The SRBCT data set has the largest number of samples with $N = 83$ and yielded the smallest average standard error of 0.0013 for the top five classification procedures. In the leukemia data set with $N = 72$, the average standard error was slightly larger at 0.0017 while the average standard error for the colon classification procedures (with $N = 62$) was 0.0069. The three simulated data sets all have $N = 40$, and interestingly the average standard error of the top five classifiers on the Sim1 and Sim3 data sets was 0.0085. The Sim2 data set had the maximum average standard error of 0.0106. The NSC procedure on the SRBCT data sets yields the minimum standard error of 0.0003 of all the procedures considered.

The standard errors of the KNN classification procedures follow the same general pattern for the data sets (besides in the Sim2 data set) and decreases as the value of K increased. Therefore, in general the 5-NN classification procedures were the most stable and had the least variation between the 200 random splits. For the SVM classification procedures, the standard error increases as the cost parameter value increased suggesting that overall the SVM with $C = 1$ is the most stable across the 200 random splits.

For the binary data sets, the SVM classifier consistently outperformed the KNN and LDA based classifiers, and the advantage of using a classification rule which is non-linear in input space, is evident.

Finally, the results of the empirical study conducted in this chapter highlight the need to fit several different base classifiers to a variety of subsets of variables from various variable selection and dimension reduction techniques. The challenge is therefore to develop a single procedure (possibly using a combination of existing procedures) that will successfully identify the relevant variables in all data sets.

In the next and final chapter, a summary of the findings in the thesis is provided, along with suggestions and recommendations for future work.

CHAPTER 7
SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

7.1 INTRODUCTION

The objective of this thesis was to investigate the performance of different classification procedures on high-dimensional microarray data sets. These procedures incorporated variable selection and/or dimension reduction to overcome the curse of dimensionality. This chapter is devoted to a summary of the main findings emanating from the research and an indication of directions for further research. In this chapter, a brief overview of the work presented in this thesis, as well as interpretations of important aspects and results from the thesis, are provided. In addition some suggestions are made regarding areas of potential interest for future research into classification procedures on high-dimensional microarray data sets.

7.2 SUMMARY

The thesis investigated different classification approaches to predict cancer subtypes using high-dimensional gene expression data. The first chapter served as an introductory chapter and provided a brief literature review of relevant papers. The next three chapters presented the theoretical background for the various statistical classification, variable selection and dimension reduction techniques implemented in this thesis (as well as a few well-known techniques that were not considered later on). The next two chapters introduced the practical data sets considered in the thesis and presented the experimental work, where practical and simulated analyses were conducted in order to compare the proposed classification procedures.

In Chapter 2, several different statistical classification techniques were discussed. The discussion focused on KNN, fastKNN, LDA, DLDA, NSC and the SVM classifier. It was pointed out that the KNN, SVM and classical LDA classifiers all suffer in high-dimensional data sets. As a way of mitigating the curse of dimensionality, variable selection and dimension reduction techniques were discussed in Chapters 3 and 4 respectively.

Variable selection and variable importance were the main topics of Chapter 3. In this chapter, a theoretical justification for VS was provided, and an overview of VS approaches which are appropriate in high-dimensional data sets was given. Two of the variable selection techniques discussed in Chapter 3, both of which fall into the category of sure independence screening, were later implemented in the empirical study, namely correlation coefficient thresholding and the two-sample t-test thresholding. Although the NSC procedure was discussed in Chapter 2 together with an explanation of the NSC classifier, the NSC procedure is the third VS procedure implemented in this thesis. The correlation VS procedure finds variables that are highly correlated with the response, but does not consider the within-class correlation between variables. As mentioned in Chapter 3, the two-sample t-test VS procedure identifies variables with a large difference of mean value between the two population groups and a small within-group variability. The NSC procedure aims to identify a subset of predictor variables which best characterises the different classes in terms of the group means.

The objective of the dimension reduction techniques (*viz.* PCA, SPCA and PLS), discussed in Chapter 4, is to reduce the number of predictors in a data set by replacing them with a much smaller number of linear transformations of the original variables. Chapter 4 discussed the benefits of using different dimension reduction techniques and included an empirical investigation focusing on a comparison of ordinary principal components with supervised principal components. The two main dimension reduction techniques implemented in this thesis were PCA and SPCA, and the first m PCs and SPCs that yield $\omega_{frac} = 0.9$ were used to "replace" the original p predictor variables.

Chapter 5 began with a detailed description of DNA microarray data sets and then presented the practical and simulated data sets considered in the thesis. The chapter then summarised the different classification procedures implemented in the empirical study of the thesis and explained how they would be implemented in R. Chapter 5 also conducted LOOCV to determine the optimal thresholding values for the correlation coefficient, two-sample t-test and NSC VS procedures. A fairly extensive empirical evaluation of the variable selection candidate values (described in Chapter 3) and the performance of KNN, SVM and NSC base classifiers (as well as the parameter values) were reported in Chapter 5. The results reported for the empirical study in Chapter 5 indicated that the link between the values of K and C used in the KNN and SVM classifiers respectively and the optimal VS thresholding value is specific to the particular data set being considered and there is no general pattern or trend evident. This highlighted that it is essential to conduct an initial empirical study on a data set in order to determine the best thresholding values for each specific VS technique combined with a particular base classifier. Overall the KNN and SVM classifiers applied to the colon cancer, Sim1 and Sim3 data sets differed more with respect to the error rates achieved at different dimensions for the correlation and t-test thresholding than when applied to the leukemia and Sim2 data sets. The KNN and SVM classifiers also behaved more erratically for the two above mentioned VS procedures on the colon, Sim1 and Sim3 data sets than on the leukemia and Sim2 data sets making it difficult to choose an optimal dimension for each classifier. In general, for the leukemia and Sim2 data sets, LOOCV chose a single candidate value as optimal for the majority of the 100 repetitions. The optimal thresholding value for both the correlation and NSC procedures reported in Section 5.5.4 were consistent for all values of K in the KNN classifier on the SRBCT data set. Therefore, it was easier to choose a dimension for the SRBCT data set as opposed to the binary data sets.

Furthermore, results reported in Chapter 5 indicated that the NSC Δ_{opt} parameter value is dependent on the classifier used. In general, for the colon cancer, leukemia and SRBCT data sets the NSC classifier selected a different Δ_{opt} value for the KNN and SVM classifiers. Finally, the NSC classifier's selection of the Δ_{opt} parameter value was concentrated over a smaller range of candidate values for all six data sets. In general the KNN and SVM classifiers' selection frequencies were sparser than those of the NSC classifier.

The optimal candidate values for the three VS techniques for $K \in \{1, 3, 5\}$ in the KNN classifier, $C \in \{1, 1\,000, 10\,000\}$ in the SVM classifier and the NSC classifier determined in Chapter 5 were then implemented in the empirical study reported in Chapter 6.

Chapter 6 was devoted to the results and interpretation of the empirical study. In summary of the above experiments, no single classifier and classification procedure stands out as the best in every application. However, the empirical study revealed that the RBF SVM outperformed the other techniques in the five binary data sets that were considered. This confirms the statement of Claeskens *et al.* (2008) that SVMs for classification have the advantage that the curse of dimensionality is circumvented. The impressive results for the SVM classifier on the binary data sets are not surprising as the SVM has previously been successfully applied to cancer classification and reported to demonstrate a high prediction accuracy. It should be borne in mind that the SVM was fine-tuned in terms of the RBF kernel hyperparameter specification.

Hastie *et al.* (2009:431) provide evidence in support of the claim that the classification accuracy of SVMs is in fact detrimentally affected by the presence of noise variables. This contradicts the findings from our empirical study for the binary data sets: using all those input variables that were available instead of only a subset of "appropriate" input variables in the RBF SVM classifier with $C \in \{1, 1\ 000, 10\ 000\}$ lead 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. Therefore, the results highlight the need to investigate other variable selection and/or dimension reduction techniques that are known to perform well with the SVM classifier.

The results for all six data sets in the empirical study indicate that the KNN classification method benefitted greatly from the initial selection of genes, which supports the results in Dudoit *et al.* (2002). Overall the KNN classification methods with $K = 5$ were more accurate than those with $K = 1$ or 3 although the differences were not always significant. For the colon, Sim1 and Sim3 data sets, the subsets of features selected by SPCA with correlation thresholding yielded the most accurate classification with the KNN classifier, while for the leukemia data set, the subset of features from SPCA with a two-sample t-test was more accurate. From these results it can be concluded that SPCA performs well with the KNN classifier for binary high-dimensional microarray data sets. However, for the SRBCT multi-class data set the NSC procedure yielded the best subset of variables for the KNN classifier.

The performance of the fastKNN classifier was very poor in comparison to that of the other classification methods considered. The results of the fastKNN feature generation procedure were also very poor, which could be attributed to the small number of neighbours used to generate the new features and consequently the very small number of features generated.

In contrast to the local KNN and fastKNN methods, LDA is a “global” method and thus made use of all the data for each prediction (Dudoit *et al.,* 2002). Consequently, some samples may not be well represented by the discriminant variables in the binary data sets. The empirical results showed that the performance of the LDA classifier on the subset of NSC shrunken genes for the binary data sets is very poor. These results are not surprising, and according to Dudoit *et al*. (2002) the most obvious reason for the poor performance of LDA is that with a limited number of samples and a fairly larger number of features, the matrices of between-class and within-class sum of squares and cross-products are quite unstable and provide poor estimates of the corresponding population quantities.

The simple DLDA rule produced impressively low misclassification rates for the three simulated data sets but performed poorly on the real colon cancer and leukemia data sets. Therefore, for the Sim1, Sim2 and Sim3 data sets considered here, gains in accuracy were achieved by ignoring the correlations between genes. The DLDA classification procedure had a remarkably low error rate when applied to 200 random splits of the leukemia data set with $p = 40$ in the paper by Dudoit *et al.* (2002).

The results showed that for the binary data sets, the DLDA classifier was the best performing linear based classifier considered in this thesis, it being superior to LDA and NSC. In general, the performance of DLDA noticeably improved when it was applied to a reduced subset of genes obtained from the NSC selection procedure. This is in line with the conclusion by Fan and Fan (2008) who theoretically studied the importance of carrying out some kind of feature selection with DLDA in high-dimensional problems. Furthermore, the results surprisingly revealed that for all the binary data sets the DLDA classifier was more accurate than the NSC classifier on the subset of NSC shrunken genes. However, the NCS classifier was the most accurate classification procedure applied to the SRBCT data set.

Finally, from the results it appears that the best variable selection and/or dimension reduction technique for high-dimensional classification is dependent on the data set and the relationships amongst and distribution of the variables in the population groups. However, the results revealed that SPCA appears to have a small advantage over PCA with the KNN classifier, conversely PCA appears to have a small advantage over SPCA with the SVM, although the differences are not significant. PLS was not implemented in this thesis. Interested readers are referred to the paper by Boulesteix (2004) which discusses and examines several aspects of PLS dimension reduction and includes an extensive study of PLS for classification methods on the colon cancer, leukemia, SRBCT and other microarray data sets.

Although the results for the top performing classification procedures on the three practical data sets compare well with those for other classification procedures from the literature, none of the procedures implemented in the thesis achieved a higher classification accuracy than existing techniques. However, it should be borne in mind that it is difficult to accurately conduct a comparative study on the practical data sets due to the different pre-processing and training-test splits used in different papers. Finally, it is reassuring to note that none of the studies to date have provided sufficient evidence in favour of a single best classifier for microarray data sets.

The investigation and interpretation of the results reported in Chapter 6 revealed that there is much scope for further research as well as further aspects that still require attention in the classification of microarray data sets. This is discussed in the next section.

7.3 RECOMMENDATIONS FOR FURTHER RESEARCH

As mentioned previously, the focus of this thesis was to compare different classification procedures on high-dimensional microarray data sets. The thesis explored several different classification procedures but there are still many directions for further research in terms of both the VS and dimension reduction techniques as well as the classification procedures.

Consider first the VS techniques. The most obvious aspect that still requires attention is that of investigating other variable selection techniques that are not based on location difference between the variables in the population groups. The three variable selection techniques investigated in this thesis (*viz.*, correlation thresholding, two-sample t-test thresholding and the NSC procedure) all use a criterion that is suitable for variable ranking in location difference cases. It is clear from the problems encountered in Section 6.2.4 that the application of location based variable selection techniques are not appropriate when applied to the Sim2 data sets which only has a scale difference between the population. Consequently, it is recommended that an investigation of and further research into alternative variable selection procedures that test for the difference in variance and distribution on a data set could be feasible. For example, the F-statistics for variable screening is one such measure. However, in practical applications it will not be known whether there is a difference in location, scale or distribution (or even a combination of the three) between the population groups. Furthermore, in certain scenarios selection measures based on a difference in distribution may be too general and may not identify any important variables. In such scenarios, the selection measures need to be more specific. Therefore, when performing VS in future applications, it is essential to consider more than one criterion for VS and not to base VS on a single selection criterion or measurement.

One issue that is important in the classification of high-dimensional microarray data sets that has not be explored in this thesis is the ability of a predictor to incorporate prior knowledge on the mRNA samples when such information is available (Dudoit *et al.*, 2002). Dudoit *et al.* (2002) highlight that in any variable selection approach one must be aware of the issue of statistical versus biological significance. A purely statistical approach may identify genes that reflect tissue sampling as opposed to biologically interesting and possibly unknown differences between the various tumours.

Further research into the use of the genetic programming classifier and mutual information based feature selection as proposed in Alladi *et al.* (2008) could be beneficial, as it has achieved a 100% accuracy on the colon data set.

Several open problems and practical limitations remain in the thesis. Due to computational time, there were practical limitations in the range of thresholding parameters considered. Furthermore, the optimal thresholding values were determined using LOOCV, and then a single pre-determined optimal value was applied to all the splits for a particular data set. However, in practice it would be beneficial to determine the optimal thresholding values based on each split of the data sets. Additionally, alternative and possibly more advanced methods for selecting the optimal threshold value (which penalizes values that retain too many variables) may improve the performance of classification using the different base classifiers on the subset of selected genes.

Now consider the scope for further research into the classification procedures implemented in this thesis. The RBF kernel was chosen as the kernel function for the SVM for the purpose of this thesis, due to its tendency to achieve positive results over other kernel functions in high-dimensional problems. However, due to the superiority of the SVM classifier on the binary data sets considered, it would be interesting to investigate the accuracy of the SVM for different kernel functions, including linear and polynomial kernel functions. Finally, due to the superiority of the SVM evident in the binary classification problem, further research should be conducted to apply the SVM classifier in a multi-class classification problem such as the SRBCT data set.

Due to the impressive performance of the SVM classifier on binary high-dimensional microarray data sets, it could be beneficial to experiment with other variable selection methods that have been found to perform well with the SVM classifier. Claeskens *et al.* (2008) provide an up-to-date review of variable selection techniques for SVMs. One such technique is the automatic gene selection method called recursive feature elimination

(RFE) proposed by Guyon *et al.* (2002). RFE starts with a discriminant function based on all the input variables and then recursively eliminates subsets of variables through optimisation of an appropriate criterion until only a pre-specified number of variables are retained. The requirement that the number of variables to retain should be pre-specified in RFE represents a serious limitation of the technique. However, Guyon *et al.* (2002) demonstrate that RFE combined with an SVM could eliminate gene redundancy automatically and yielded better and more compact gene subsets.

Additionally, extending the fastKNN procedure to use larger values of K would seem to be a feasible undertaking and would provide more insight into the procedure.

REFERENCES

Afifi, A.A. and Clark, V. 1997. *Computer-Aided Multivariate Analysis*, 3rd Edition. Florida: Chapman & Hall/ CRC.

Alladi, S.M., Shinde Santosh, P., Ravi, V. and Murthy, U.S. 2008. Colon cancer prediction with genetic profiles using intelligent techniques. *Bioinformation*, **3**(3), 130-133.

Alon, U., Barkai, N. Notterman, D. A., Gish, K., Ybarra, S., Mack, D. and Levine, A. J. 1999. Broad Patterns of Gene Expression Revealed by Clustering Analysis of Tumor and Normal Colon Tissues Probed by Oligonucleotide Arrays. *Proc. Natl. Acad. Sci. USA*, Vol. 96, Issue 12, 6745-6750.

Ambroise, C. and McLachlan, G. 2002. Selection bias in gene extraction on the basis of microarray gene-expression data*, Proceedings of the National Academy of Sciences* **99**: 6562–6566.

American Cancer Society. 2014. *Understanding Your Pathology Report: Invasive Adenocarcinoma of the Colon*. [Online] Available: http://www.cancer.org/treatment/understandingyourdiagnosis/understandingyourpathologyreport/colonpathology/invasive-adenocarcinoma-of-the-colon [2017, March 19].

Antoniadis, A., Lambert-Lacroix, S. and Leblanc, F. 2003. Effective dimension reduction methods for tumor classification using gene expression data. *Bioinformatics*, **19**(5), 563-570.

Babu, M.M. 2004. Introduction to microarray data analysis. *Computational genomics: Theory and application*, **17**(6), 225-249.

Bair, E. and Tibshirani, R. 2004. Semi-supervised methods to predict patient survival from gene expression data. *PLoS biology*, 2(4), 511-522.

Bair, E., Hastie, T., Paul, D. and Tibshirani, R. 2006. Prediction by supervised principal components. *Journal of the American Statistical Association*, **101**, 119-137.

Ben-Dor, A., Bruhn, L., Friedman, N., Nachman, I., Schummer, M. and Yakhini, Z. 2000. Tissue classification with gene expression profiles. *Journal of computational biology*, 7(3-4), 559-583.

Benjamini, Y. and Hochberg, Y. 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B*, **85**, 289-300.

Bickel, P.J. and Levina, E. 2004. Some theory for Fisher's linear discriminant function, “Naive Bayes”, and some alternatives when there are many more variables than observations. *Bernoulli,* **10**, 989-1010.

Bierman, S. and Steel, S. 2009. Variable selection for support vector machines. *Communications in Statistics-Simulation and Computation*, 38(8), 1640-1658.

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

Boulesteix, A.L. 2004. PLS dimension reduction for classification with microarray data. *Statistical applications in genetics and molecular biology*, **3**(1), 1075-1105.

Buhler, J. 2002. Anatomy of a Comparative Gene Expression Study. [Online] Available: http://www.cs.wustl.edu/~jbuhler/research/array/ [2017, February 19].

Campa, M.J., Wang, M.Z., Howard, B., Fitzgerald, M.C. and Patz, E.F. 2003. Protein expression profiling identifies macrophage migration inhibitory factor and cyclophilin a as potential molecular targets in non-small cell lung cancer. *Cancer research, **63***(7), 1652-1656.

Cherian, M.G., Jayasurya, A. and Bay, B.H. 2003. Metallothioneins in human tumors and potential roles in carcinogenesis. *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis, **533***(1), 201-209.

Chow, M.L., Moler, E.J. and Mian, I.S. 2001. Identifying marker genes in transcription profiling data using a mixture of feature relevance experts. *Physiological genomics, **5**(2),* 99-111.

Chun, H. and Keleş, S. 2010. Sparse partial least squares regression for simultaneous dimension reduction and variable selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology),* **72**, 3-25.

Chung, D. and Keleş, S. 2010. Sparse Partial Least Squares Classification for High Dimensional Data. *Statistical Applications in Genetics and Molecular Biology*, Vol. 9, Article 17.

Claeskens, G., Croux, C., Van Kerkhoven, J. 2008. An information criterion for variable selection in support vector machines. *Journal of Machine Learning Research,* 9:541–558.

Dasarathy, B. 1991. *Nearest Neighbor Pattern Classification Techniques*. Los Alamitos, CA: IEEE Computer Society Press.

Dettling, M. and Bühlmann, P. 2002. Supervised clustering of genes. *Genome Biology*, 3: 1–15.

Díaz-Uriarte, R. and Alvarez de Andrés, S. 2006. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, **7**, 3.

Donoho, D. and Johnstone, I. 1994. Ideal spatial adaptation by wavelet shrinkage. *Biometrika,* **81,** 425–455.

Dudoit, S., Fridlyand, J. and Speed, T. 2002. Comparison of discrimination methods for the classification of tumors using gene expression data. *Journal of the American Statistical Association,* **97**(457), 77–87.

Efron, B. 2009. Empirical Bayes estimates for large-scale prediction problems. *Journal of the American Statistical Association*, **104**(487), 1015-1028.

Efron, B., Hastie, T., Johnstone, I. and Tibshirani, R. 2004. Least angle regression (with discussion). *The Annals of Statistics,* **32**(2): 407-409.

Fan, J and Fan, Y. 2008. High-dimensional classification using features annealed independence rules. *The Annals of Statistics*, **36**, 2605–2637.

Fan, J. and Lv, J. 2010. A selective overview of variable selection in high dimensional feature space. *Statistica Sinica*, **20**, 101–148.

Fan, J. and Lv, J. 2008. Sure independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **70**(5), 849-911.

Fischer, H., Sterling, R., Rubio, C., & Lindblom, A. 2001. Colorectal carcinogenesis is associated with stromal expression of *COL11A1* and *COL5A2*. *Carcinogenesis, 22(6)*, 875–878.

Fisher, R.A. 1936. The use of multiple measurements in taxonomic problems. *Annals of human genetics*. **7**: 179-188.

Fix, E. and Hodges, J. 1951. Discriminatory analysis—nonparametric discrimination: Consistency properties, *Technical Report 21-49-004,4,* U.S. Air Force, School of Aviation Medicine, Randolph Field, TX.

Ghosh, D. 2002. Singular value decomposition regression models for classification of tumors from microarray experiments. *Proceedings of the 2002 Pacific Symposium on Biocomputing*. 11462-11467.

Golub, T., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J., Caligiuri, M. A., Bloomfield, C. D. and Lander, E. S. 1999. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science,* **286**, 531–537.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. 2002. Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning.* **46**, 1-3.

Hastie, T., Tibshirani, R. and Friedman, J. 2009. *The elements of statistical learning*. Second edition. New York: Springer.

Hesterberg, T., Choi, N.H., Meier, L. and Fraley, C. 2008. Least angle and ℓ_1 penalized regression: A review*. Statistics Surveys*, 2, 61-93.

Howard Hughes Medical Institute. 2017. How to Analyze DNA Microarray Data. [Online] Available: http://www.hhmi.org/biointeractive/how-analyze-dna-microarray-data [2017, February 20].

James, G., Witten, D., Hastie, T., Tibshirani, R. 2013. *An Introduction to Statistical Learning: with applications in R*. New York: Spring.

Jiang, W., Li, X., Rao, S., Wang, L., Du, L., Li, C., Wu, C., Wang, H., Wang, Y. and Yang, B. 2008. Constructing disease-specific gene networks using pair-wise relevance metric: application to colon cancer identifies interleukin 8, desmin and enolase 1 as the central elements. *BMC systems biology*, 2(1), 72.

Johnson, R.A. and Wichern, D.W. 2007. *Applied Multivariate Statistical Analysis.* New Jersey: Pearson Education Limited.

Jolliffe, I. 1986. *Principal Component Analysis*, New York: Springer Verlag.

Kahn, J., Wei, J. S., Ringner, M., Saal, L. H., Ladanyi, M., Westermann, F., Berthold, F., Schwab, M., Antonescu, C. R., Peterson, C., Meltzer, P. S. 2001. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature Medicine* 7, 673–679.

Keely, P., Parise, L., and Juliano, R. 1998. Integrins and GTPases in tumour cell growth, motility and invasion. *Trends in cell biology*, *8*(3):101–107.

Kishino, H. and Waddell, P.J. 2000. Correspondence analysis of genes and tissue types and finding genetic links from microarray data. *Genome Informatics*, 11, 83-95.

Kulkarni, A., Kumar, B.N., Ravi, V. and Murthy, U.S. 2011. Colon cancer prediction with genetics profiles using evolutionary techniques. *Expert Systems with Applications*, **38**(3), 2752-2757.

Lee, M.L.T. 2004. *Analysis of Microarray Gene Expression Data*. Boston: Kluwer Academic Publishers.

Levina, E. 2002. *Statistical issues in texture analysis* (Doctoral dissertation, University of California, Berkeley).

Li, X. and Xu, R. (Ed.). 2008. *High-dimensional data analysis in cancer research*. Springer: New York.

Liu, H., Wasserman, L. and Lafferty, J.D. 2009. Nonparametric regression and classification with joint sparsity constraints. *Advances in neural information processing systems,* 969-976.

Louw, N. 1997. *Aspects of the pre- and post-selection classification performance of discriminant analysis and logistic regression. PhD Thesis*, The Department of Statistics and Actuarial Science, University of Stellenbosch, South Africa.

Lu, F. and Petkova, E. 2014. A comparative study of variable selection methods in the context of developing psychiatric screening instruments. *Statistics in Medicine,* **33**, 401-421.

Mardia, K., Kent, J. and Bibby, J. 1979. *Multivariate Analysis*. New York: Academic Press.

McLachlan, G. 1992. *Discriminant Analysis and Statistical Pattern Recognition*. New York: Wiley.

Notterman, D.A., Alon, U., Sierk, A.J. and Levine, A.J. 2001. Transcriptional gene expression profiles of colorectal adenoma, adenocarcinoma, and normal tissue examined by oligonucleotide arrays. *Cancer research*, **61**(7), 3124-3130.

Oxford Dictionaries. 2017. [Online]. Available: https://en.oxforddictionaries.com/definition/dna. [2017, February 19].

Parmigiani, G., Garrett, E.S., Irizarry, R.A. and Zeger, S.L. (Ed.). 2003. *The analysis of gene expression data: an overview of methods and software*. New York: Springer.

Paul, D., Bair, E., Hastie, T. and Tibshirani, R. 2008. "Preconditioning" for feature selection and regression in high-dimensional problems. *The Annals of Statistics*, **36**, 1595-1618.

Pinto, D. 2016. Introduction to fastknn [Online]. Available: https://github.com/davpinto/fastknn [2017, February 8].

Poplack, D.G. and Pizzo, D.G. 1997. *Principles and practice of pediatric oncology*. Third Edition. Lippincott-raven: Hagerstown MD.

Rakotomamonjy, A. 2003. Variable Selection Using SVM-based Criteria. *Journal of Machine Learning Research.* **3**, 1357-1370.

Ramaswamy, S., Tamayo, P., Rifkin, R., Mukherjee, S., Yeang, C.H., Angelo, M., Ladd, C., Reich, M., Latulippe, E., Mesirov, J.P. and Poggio, T. 2001. Multiclass cancer diagnosis using tumor gene expression signatures. *Proceedings of the National Academy of Sciences*, **98**(26), 15149-15154.

Rao, C.R. 1973. *Linear Statistical Inference and Its Applications*. New York: Wiley.

Rencher, A. C. 2002. *Methods of Multivariate Analysis. Second Edition.* New York: Wiley.

Ripley, B. D. 1996. *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press.

Schëlkopf, B. and Smola, A.I. 2002. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. Cambridge, MA: MIT Press.

Shailubhai, K., Helen, H.Y., Karunanandaa, K., Wang, J.Y., Eber, S.L., Wang, Y., Joo, N.S., Kim, H.D., Miedema, B.W., Abbas, S.Z. and Boddupalli, S.S. 2000. Uroguanylin treatment suppresses polyp formation in the ApcMin/+ mouse and induces apoptosis in human colon adenocarcinoma cells via cyclic GMP. *Cancer research*, 60(18), 5151-5157.

Simon, R. M., Korn, E. L., McShane, L. M., Radmacher, M. D., Wright, G. and Zhao, Y. 2004. *Design and Analysis of DNA Microarray Investigations*. New York: Springer.

Speed, T. (Ed.). 2003. *Statistical Analysis of Gene Expression Microarray Data*. London: Chapman and Hall.

Steel, S.J., Louw, N. and Bierman, S. 2011. Variable Selection for Kernel Classification. *Communications in Statistics—Simulation and Computation, 40:2*, 229-246.

Tibshirani, R. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58, 267–288.

Tibshirani, R., Hastie, T., Narasimhan, B. and Chu, G. 2001. Diagnosis of multiple cancer types by shrunken centroids of gene expression, *Proceedings of the National Academy of Sciences,* **99**, 6567–6572.

Tibshirani, R., Hastie, T., Narasimhan, B. and Chu, G. 2003. Class prediction by nearest shrunken centroids, with applications to DNA microarrays. *Statistical Science*, **18**(1), 104–117.

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

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

Wang, Z., Gerstein, M. and Snyder, M. 2009. RNA-Seq: a revolutionary tool for transcriptomics. *Nature reviews genetics*, **10**(1), 57-63.

Weston, J., Elisseff, A., Schölkopf, B., and Tipping, M. 2003. Use of the Zero-Norm with Linear Models and Kernel Methods. *Journal of Machine Learning Research,* **3**, 1439-1461.

Wold, H. 1966. Estimation of Principal Components and Related Models by Iterative Least Squares. *In Multivariate Analysis, Krishnaiah, P (ed.)*. New York: Academic Press.

Wold, H. 1975. Soft modelling by latent variables: the nonlinear iterative partial least squares (NIPALS) approach, *Perspectives in Probability and Statistics, In Honor of M. S. Bartlett*. 117–144.

Yam, J., Chan, K., and Hsiao, W. 2001. Suppression of the tumorigenicity of mutant p53- transformed rat embryo fibroblasts through expression of a newly cloned rat nonmuscle myosin heavy chain-B. *Oncogene, 20(1),* 58–68.

Yap, Y., Zhang, X., Ling, M.T., Wang, X., Wong, Y.C. and Danchin, A. 2004. Classification between normal and tumor tissues based on the pair-wise gene expression ratio. *BMC cancer*, *4*(1), 72.

Ye, J. and Li, Q. 2005. A two-stage linear discriminant analysis via QR-decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **27**, 929-41.

Zhang, S. and Sim, T. 2007. Discriminant subspace analysis: a Fukunaga-Koontz approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* **29**, 1732-1745.

Zhao, S., Fung-Leung, W.P., Bittner, A., Ngo, K. and Liu, X. 2014. Comparison of RNA-Seq and microarray in transcriptome profiling of activated T cells*. PloS one*, *9*(1), 1:13.

APPENDIX A:
COMPREHENSIVE EMPIRICAL STUDY RESULTS

Table A.1: Ranking of the classification procedures on the four binary data sets

		Test Error rankings for				
Rank	**Method**	**Colon**	**Leukemia**	**Sim1**	**Sim3**	**All four**
1	SVM (C =1 000)	7	1	17	9	8.50
2	cor-SVM ($C = 1$)	10	3	13	11	9.25
3	SVM ($C = 10\ 000$)	9	2	16	10	9.25
4	cor-SVM ($C = 1\ 000$)	8	4	14	12	9.50
5	cor-SVM ($C = 10\ 000$)	11	5	15	13	11.00
6	ttest-SVM ($C = 1$)	1	12	7	33	13.25
7	cor-SPCA-KNN ($K = 5$)	2	15	20	17	13.50
8	NSC-DLDA	12	25	2	15	13.50
9	SVM ($C = 1$)	41	6	10	4	15.25
10	cor-KNN ($K = 5$)	5	17	26	14	15.50
11	PCA-SVM ($C = 1\ 000$)	29	21	12	2	16.00
12	PCA-SVM ($C = 10\ 000$)	28	22	11	3	16.00
13	PCA-SVM ($C = 1$)	16	55	1	1	18.25
14	NSC-SVM ($C = 1$)	13	36	3	24	19.00
15	cor-SPCA-KNN ($K = 3$)	4	34	22	21	20.25
16	cor-KNN ($K = 3$)	3	27	33	22	21.25
17	ttest-SPCA-KNN ($K = 5$)	19	8	40	19	21.50
18	ttest-KNN ($K = 5$)	17	10	37	23	21.75
19	cor-SPCA-SVM ($C = 1\ 000$)	34	19	29	6	22.00
20	cor-SPCA-SVM ($C = 10\ 000$)	33	20	27	8	22.00
21	DLDA	54	23	6	5	22.00
22	ttest-SPCA-KNN ($K = 3$)	22	7	36	25	22.50
23	ttest-SVM ($C = 10\ 000$)	27	16	9	39	22.75
24	ttest-SVM ($C = 1\ 000$)	30	14	8	40	23.00
25	cor-SPCA-SVM ($C = 1$)	6	54	28	7	23.75
26	NSC-KNN ($K = 5$)	14	39	30	16	24.75
27	ttest-KNN ($K = 3$)	20	9	43	27	24.75

		Test Error rankings for				
Rank	**Method**	**Colon**	**Leukemia**	**Sim1**	**Sim3**	**All four**
28	cor-fastKNN ($K = 5$)	36	18	18	35	26.75
29	NSC-KNN ($K = 3$)	21	38	32	20	27.75
30	NSC	23	45	19	26	28.25
31	NSC-SVM ($C = 1\,000$)	44	30	4	36	28.50
32	NSC-SVM ($C = 10\,000$)	42	29	5	38	28.50
33	KNN ($K = 3$)	15	26	46	28	28.75
34	KNN ($K = 5$)	25	33	44	18	30.00
35	cor-SPCA-KNN ($K = 1$)	35	24	34	29	30.50
36	ttest-SPCA-SVM ($C = 1$)	18	53	23	32	31.50
37	cor-KNN ($K = 1$)	37	28	35	31	32.75
38	ttest-SPCA-KNN ($K = 1$)	40	11	41	41	33.25
39	ttest-KNN ($K = 1$)	38	13	42	46	34.75
40	ttest-SPCA-SVM ($C = 10\,000$)	45	32	25	43	36.25
41	ttest-SPCA-SVM ($C = 1\,000$)	47	31	24	45	36.75
42	cor-fastKNN ($K = 3$)	46	44	31	30	37.75
43	NSC-KNN ($K = 1$)	43	37	38	37	38.75
44	PCA-KNN ($K = 5$)	24	35	48	49	39.00
45	fastKNN ($K = 5$)	39	46	47	34	41.50
46	PCA-KNN ($K = 3$)	26	41	50	52	42.25
47	KNN ($K = 1$)	31	42	49	50	43.00
48	NSC-LDA	48	48	21	55	43.00
49	PCA-KNN ($K = 1$)	32	40	51	54	44.25
50	fastKNN-SVM ($C = 1$)	49	43	45	42	44.75
51	cor-fastKNN ($K = 1$)	55	47	39	44	46.25
52	fastKNN-SVM ($C = 1\,000$)	50	51	53	47	50.25
53	fastKNN-SVM ($C = 10\,000$)	51	50	52	48	50.25
54	fastKNN ($K = 3$)	52	49	54	51	51.50
55	fastKNN ($K = 1$)	53	52	55	53	53.25

Table A.2: Ranking of the classification procedures on the Sim2 data set

Rank	Method	Number of features	Test Error Rate	Std. Error Rate
1	SVM ($C = 1$)	1 000	0.3994	0.0092
2	DLDA	1 000	0.4275	0.0108
3	fastKNN ($K = 5$)	10	0.4313	0.0112
4	SVM ($C = 1\ 000$)	1 000	0.4331	0.0110
5	SVM ($C = 10\ 000$)	1 000	0.4331	0.0110
6	fastKNN-SVM ($C = 1$)	10	0.4356	0.0099
7	fastKNN ($K = 3$)	6	0.4456	0.0117
8	KNN ($K = 3$)	1 000	0.4625	0.0112
9	PCA-SVM ($C = 1\ 000$)	27	0.4688	0.0108
10	PCA-SVM ($C = 10\ 000$)	27	0.4688	0.0108
11	fastKNN-SVM ($C = 1\ 000$)	10	0.4725	0.0110
12	fastKNN-SVM ($C = 10\ 000$)	10	0.4725	0.0115
13	fastKNN ($K = 1$)	2	0.4831	0.0115
14	PCA-SVM ($C = 1$)	27	0.4869	0.0106
15	KNN ($K = 5$)	1 000	0.4900	0.0119
16	PCA-KNN ($K = 1$)	27	0.5244	0.0103
17	PCA-KNN ($K = 3$)	27	0.5331	0.0103
18	PCA-KNN ($K = 5$)	27	0.5331	0.0114
19	KNN ($K = 1$)	1 000	0.5500	0.0112

Table A.3: Ranking of the classification procedures on the SRBCT data set

Rank	Method	No. of features	Test Error Rate	Std. Error
1	NSC	48	0.0003	0.0003
2	NSC-KNN ($K = 1$)	48	0.0006	0.0004
3	NSC-KNN ($K = 5$)	48	0.0008	0.0005
4	NSC-KNN ($K = 3$)	48	0.0025	0.0009
5	KNN ($K = 3$)	2 308	0.0933	0.0045
6	KNN ($K = 5$)	2 308	0.1039	0.0049
7	PCA-KNN ($K = 3$)	35	0.1064	0.0045
8	fastKNN ($K = 5$)	20	0.1111	0.0046
9	KNN ($K = 1$)	2 308	0.1150	0.0050
10	PCA-KNN ($K = 5$)	35	0.1300	0.0053
11	PCA-KNN ($K = 1$) ($K = 1$)	35	0.1383	0.0058
12	fastKNN ($K = 3$)	12	0.1464	0.0049
13	fastKNN ($K = 1$)	4	0.1742	0.0059
14	cor-KNN ($K = 1$)	7	0.2722	0.0072
15	cor-KNN ($K = 3$)	7	0.3053	0.0066
16	cor-SPCA-KNN	4	0.3125	0.0079
17	cor-KNN ($K = 5$)	7	0.3144	0.0066
18	cor-SPCA-KNN ($K = 3$)	4	0.3258	0.0078
19	cor-SPCA-KNN ($K = 5$)	4	0.3306	0.0067
20	cor-fastKNN ($K = 3$)	12	0.3453	0.0071
21	cor-fastKNN ($K = 5$)	20	0.3475	0.0069
22	cor-fastKNN ($K = 1$)	4	0.3503	0.0073

APPENDIX B:
R PROGRAMMES

B.1 Packages to install In R

```
set.seed(1)
############################################################################
#                                                                         ##
# Loading the necessary packages and libraries in R                       ##
#                                                                         ##
############################################################################
library(MASS)      #mvrnorm function
library(class)
library(Matrix)
library(leaps)     #regsubset function
library(stats)     #for Pcor
library(caret)     #contains the KNN classifier
library(varbvs)    #contains the leukemia data set
library(RANN)
library(foreach)
library(Metrics)
library(matrixStats)
library(ggplot2)
library(viridis)
library(devtools)
library(fastknn)   #contains the fastKNN classifier
library(pamr)      #NSC package
library(kernlab)
```

B.2 Importing the Colon Cancer Data set into R

```
> fix(Pcancerdata)
function () {
############################################################################
#                                                                         ##
# This program imports the colon cancer data from an excel file           ##
#                                                                         ##
############################################################################
 cmat1=matrix(scan("C:\\Users\\Tess\\Documents\\2016-2017 Masters\\
                  Thesis\\Data\\colon.d"),nrow=62,ncol=2000)
 yvec=matrix(scan("C:\\Users\\Tess\\Documents\\2016-2017 Masters\\
             Thesis\\Data\\colonlabels.d"),nrow=62,ncol=1)
 yvec1=as.numeric(yvec<0)   #changing y into a vec of 0's and 1's
 cmat2=cbind(cmat1,yvec1)

## Transforming the y-variable into 0's & 1's##
 ind1=which(yvec1==0)    #Group 1: Normal Tissue
 ind2=which(yvec1==1)    #Group 2: Tumerous Tissue

############################################################################
#                                                                         ##
# Outputting cancer.xy that will be used in the thesis                    ##
#                                                                         ##
############################################################################
 cancer.xy=rbind(cmat2[ind1,],cmat2[ind2,])
 return(cancer.xy)
}
```

B.3 Loading the pre-processed Leukemia data set into R

```
> fix(Pleukemiadata)
function () {
##########################################################################
#                                                                       ##
# This program loads/ inputs the leukemia data from the varbvs library ##
#                                                                       ##
##########################################################################
  library(varbvs)
  data(leukemia)

## Outputting leukemia.xy data set that will be used in the thesis ##
 leukemia.xy = cbind(leukemia$x, leukemia$y)
 return(leukemia.xy)
}
```

B.4 Importing the SRBCT Data set into R

```
> fix(Psrbctdata)
function () {
##########################################################################
#                                                                       ##
# This program imports the srbct data set from:                         ##
#  https://statweb.stanford.edu/~tibs/ElemStatLearn/data.html           ##
#                                                                       ##
##########################################################################
## Importing the test data ##
    srbct.testx= t(read.table("https://statweb.stanford.edu/~tibs/
                   ElemStatLearn/datasets/khan.xtest", header=F))
    srbct.testy=t(read.table("https://statweb.stanford.edu/~tibs/
                   ElemStatLearn/datasets/khan.ytest", header=F))

## Finding NA value's & removing them to create testxy data ##
    na.index=which(is.na(srbct.testy))              #checking for NA's
    srbct.testxy = cbind(srbct.testx[-na.index,],srbct.testy[-na.index])

## Importing Training data ##
    srbct.trainx = t(read.table("https://statweb.stanford.edu/~tibs/
                    ElemStatLearn/datasets/khan.xtrain", header=F))
    srbct.trainy = t(read.table("https://statweb.stanford.edu/~tibs/
                     ElemStatLearn/datasets/khan.ytrain", header=F))

    srbct.trainxy = cbind(srbct.trainx,srbct.trainy)

##########################################################################
#                                                                       ##
# Outputting srbct.xy that will be used in the thesis                   ##
#                                                                       ##
##########################################################################
   srbct.xy = rbind(srbct.trainxy,srbct.testxy)
   return(srbct.xy)
}
```

B.5 Generating the Sim1 data set

```
>  fix(PSim1)
function (p, prel, N1, N2) {
   # p = number of predictor variables
   # prel = number of relevant predictor variables
   # N1 & N2 number of observations in Group 1 & Group 2 respectively

#########################################################################
#                                                                      ##
# This programme simulates binary microarray data by adding            ##
#  a location difference between the 2 population groups              ##
#                                                                      ##
#########################################################################

 N = N1 + N2
 xmat = matrix(rnorm(p*N,0,1), ncol=p, nrow=N)

 for (i in (N1+1):N){   xmat[i, 1:prel] = xmat[i, 1:prel] + 0.5  }
 yvec = c(rep(0,N1), rep(1,N2))
 data.xy = cbind(xmat, yvec)

 return(data.xy)
}
```

B.6 Generating the Sim2 data set

```
>  fix(PSim2)
function (p, prel, N1, N2) {
   # p = number of predictor variables
   # prel = number of relevant predictor variables
   # N1 & N2 number of observations in Group 1 & Group 2 respectively

#########################################################################
#                                                                      ##
# This programme simulates binary microarray data by adding            ##
#   a scale variation of 0.5² between the 2 population groups         ##
#                                                                      ##
#########################################################################

 N = N1 + N2
 xmat = matrix(rnorm(p*N,0,1), ncol=p, nrow=N)

 for (i in (N1+1):N){ xmat[i,1:prel] = xmat[i,1:prel] + rnorm(1,0,0.5)}
 yvec = c(rep(0,N1), rep(1,N2))
 data.xy = cbind(xmat, yvec)

 return(data.xy)
}
```

B.7 Generating the Sim3 data set

```
> fix(PSim3)
function (p, prel, N1, N2) {
   # p = number of predictor variables
   # prel = number of relevant predictor variables
   # N1 & N2 number of observations in Group 1 & Group 2 respectively

#########################################################################
#                                                                      ##
# This programme simulates binary microarray data by adding            ##
#   a location difference & scale variation between the 2 groups       ##
#                                                                      ##
#########################################################################

 N = N1 + N2
 xmat = matrix(rnorm(p*N,0,1), ncol=p, nrow=N)

 for (i in (N1+1):N){xmat[i,1:prel]= xmat[i,1:prel] + rnorm(1,0.5,0.5) }
 yvec = c(rep(0,N1), rep(1,N2))
 data.xy = cbind(xmat, yvec)

 return(data.xy)
}
```

B.8 Generating the new features from fastKNN in a binary data set

```
> fix(PfastKNN)
function (traindata.x, traindata.y, K, testdata.x) {
#########################################################################
#                                                                      ##
# This programme implements fastKNN feature engineering in binary data
##
#                                                                      ##
#########################################################################

 ## Splitting the training index according to the two groups ##
  G =length(unique(traindata.y))
  N = nrow(traindata.x)
  ind0 = which(traindata.y == 0)
  ind1 = which(traindata.y == 1)
  N0 = length(ind0)
  N1 = length(ind1)

 ## For each group calculate ##
  new.feat = rep(0,K*G)

  for (g in 1:G) {

   #a) distance to test observation #
         if(g==1) {
              ind=ind0
              dist=rep(0,N0) }
          if (g==2) {
              ind=ind1
              dist=rep(0,N1)}

        for (i in ind)  {
            j = which(ind==i)
```

```
             dist[j] = sqrt(sum((testdata.x - traindata.x[i,])^2))    }

  #b) order the distances #
           sortdist = sort(dist,decreasing=F, index.return=T)

  #c) Pick the K smallest distances #
          indk = sortdist$ix[1:K]

  #d) Compute the K new features #
           for (k in 1:K)
           new.feat[(g-1)*K +k] = sum(dist[indk[1:k]])
    }
 return(new.feat)
}
```

B.9 Performing LOOCV in fastKNN in a binary data set

```
> fix(PfastKNNset)
function (traindata.x, traindata.y, K, testdata.x) {
#########################################################################
#                                                                      ##
# This programme performs LOOCV for each new training set to avoid     ##
#  overfitting when implementing fastKNN in binary data set            ##
#                                                                      ##
#########################################################################
## Computing the number of observations in the training and test data ##
   Ntrain = nrow(traindata.x)
   Ntest = nrow(testdata.x)
   G =length(unique(traindata.y))
   new.feat.tr = matrix(0, ncol=K*G, nrow=Ntrain)
   new.feat.te = matrix(0, ncol=K*G, nrow=Ntest)

 ##  Performing LOOCV ##
   for (i in 1:Ntrain) new.feat.tr[i,]=PfastKNN(traindata.x[-i,],
                                    traindata.y[-i], K, traindata.x[i,])

   for (i in 1:Ntest) new.feat.te[i,]=PfastKNN(traindata.x, traindata.y,
                                              K, testdata.x[i,])

  return(list("new.feat.tr"=new.feat.tr, "new.feat.te"=new.feat.te))
}
```

B.10 Generating the new features from fastKNN in a multiclass data set

```
> fix(PfastKNN_mult)
function (traindata.x, traindata.y, K, testdata.x) {
##########################################################################
#                                                                       ##
# This programme implements fastKNN feature engineering in binary data
##
#                                                                       ##
##########################################################################

## Splitting the training ind according to the groups ##
  G =length(unique(traindata.y))
  N = nrow(traindata.x)

  ind1 = which(traindata.y == 1)
  ind2 = which(traindata.y == 2)
  ind3 = which(traindata.y == 3)
  ind4 = which(traindata.y == 4)
  N1  = length(ind1)
  N2  = length(ind2)
  N3  = length(ind3)
  N4  = length(ind4)

 ## For each group calculate ##
  new.feat  = rep(0,K*G)
  for  (g in 1:G)  {

  #a)  distance to test observation #
     if(g==1)  {
        ind=ind1
        dist=rep(0,N1)  }

     if  (g==2)  {
         ind=ind2
         dist=rep(0,N2)}

     if  (g==3)  {
         ind=ind3
         dist=rep(0,N3)}

     if  (g==4)  {
         ind=ind4
         dist=rep(0,N4)}

   for  (i  in ind)   {
       j  = which(ind==i)
       dist[j]  = sqrt(sum((testdata.x - traindata.x[i,])^2))    }

  #b)  order the distances #
      sortdist  = sort(dist,decreasing=F,  index.return=T)

  #c)  Pick the K smallest distances #
       indk  = sortdist$ix[1:K]

  #d)  Compute the K new features #
       for  (k in 1:K)
         new.feat[(g-1)*K +k]  =  sum(dist[indk[1:k]])
   }
 return(new.feat)
}
```

B.11 Performing LOOCV in fastKNN in a multi-class data set

```
>  fix(PfastKNNset)
function (traindata.x, traindata.y, K, testdata.x) {
#########################################################################
#                                                                      ##
# This programme implements LOOCV on each new training set to avoid    ##
#  overfitting when implementing fastKNN in multi-class data set       ##
#                                                                      ##
#########################################################################
## Computing the number of obs in the training and test data ##
   Ntrain = nrow(traindata.x)
   Ntest = nrow(testdata.x)
   G = 4
   new.feat.tr = matrix(0, ncol=K*G, nrow=Ntrain)
   new.feat.te = matrix(0, ncol=K*G, nrow=Ntest)

 ##  Performing LOOCV ##

   for (i in 1:Ntrain) new.feat.tr[i,]= PfastKNN_mult(traindata.x[-i,],
                      traindata.y[-i], K, traindata.x[i,])

   for (i in 1:Ntest) new.feat.te[i, = PfastKNN_mult(traindata.x,
                      traindata.y, K, testdata.x[i,])

  return(list("new.feat.tr"=new.feat.tr, "new.feat.te"=new.feat.te))

}
```

B.12 Implementing LDA on the binary data as in Section 2.4

```
>  fix(PLDA)
function (traindata.x,traindata.y,testdata.x) {
#########################################################################
#                                                                      ##
# This program implements LDA on the training and test data set        ##
#                                                                      ##
#########################################################################

## Data properties ##
 xmat = traindata.x
 ymat=cbind((1-traindata.y),traindata.y)
 G=ngroups=ncol(ymat)                          #No. of population groups
 xmat1 = xmat[traindata.y==0,]
 xmat2 = xmat[traindata.y==1,]
 N1 = nrow(xmat1)
 N2 = nrow(xmat2)
 N = N1+N2                           #No. of obs
 nofeat = ncol(xmat)                 #No. of features used (not always p)

#########################################################################
#                                                                      ##
# Classical LDA calculations                                           ##
#                                                                      ##
#########################################################################
 ## Obtaining the Sw, Sb matrices and their precursor matrices Hb & Hw ##
   hbmat=matrix(0,nrow=nofeat,ncol=G)      #Creating a nofeat x G matrix
   ysum=apply(ymat,2,sum)                  #Computing Ni
   vec1_N=rep(1,N)                          #vector of 1's of length N
```

```
    vec1_G=rep(1,G)                                 #vector of 1's of length G
    grmeanmat= t(xmat)%*%ymat%*%diag(1/ysum)       #xi.bar = group mean
    meanmat= (grmeanmat%*%t(ymat)%*%vec1_N%*%t(vec1_G))/N  #x̄= overall mean
    hbmat = ((grmeanmat-meanmat)%*%diag(sqrt(ysum)))/sqrt(N)
    sbmat = hbmat%*%t(hbmat)

    hwmat = (t(xmat)-grmeanmat%*%t(ymat))/sqrt(N)
    swmat = hwmat%*%t(hwmat)
    stmat = sbmat+swmat        #St= Sb + Sw

    mat1= solve(swmat+0.000000001*diag(nrow(swmat)))%*%sbmat   #Sw(-1)Sb
    kk1 = as.double((eigen(mat1))$vector[,1:(G-1)])
      ##kk1=eigenvalues of Sw(-1)Sb, as.double discards the imaginery parts

    w.vec = matrix(kk1,nrow=nofeat)          #w projection vector
    m = t(grmeanmat)%*%w.vec                 # closest mean classifier

###########################################################################
#                                                                        ##
# Computing and outputting the classification                            ##
#                                                                        ##
###########################################################################
   yvec.predict = NULL

   for (i in 1:nrow(testdata.x)) {
    y0 = testdata.x[i,]%*%w.vec
    dist = abs(rep(y0,2)- m)
    yvec.predict[i] = which.min(dist)-1
   }
  return(yvec.predict)
 }
```

B.13 Implementing DLDA on the binary data as in Section 2.4.1

```
>  fix(PdiagLDA)
function (traindata.x,traindata.y,testdata.x) {
###########################################################################
#                                                                        ##
# This program implements DLDA on the training and test data set         ##
#                                                                        ##
###########################################################################

## Data properties ##
 xmat = traindata.x
 G =length(unique(traindata.y))
 xmat1 = xmat[traindata.y==0,]
 xmat2 = xmat[traindata.y==1,]
 N1 = nrow(xmat1)
 N2 = nrow(xmat2)
 N = N1+N2                    #number of observations
 nofeat = ncol(xmat)        #number of features used (not always p)

###########################################################################
#                                                                        ##
# Computing the DLDA discriminant score for the 2 classes as in (2.3)    ##
#  and outputting the classification                                     ##
#                                                                        ##
###########################################################################
```

```
    meanmat = matrix(0,nrow=2,ncol=nofeat)
    meanmat[1,] = apply(xmat1,2,mean)
    meanmat[2,] = apply(xmat2,2,mean)

    svec = NULL
     for (j in 1:nofeat) {
      term1= sum((xmat1[,j] - meanmat[1,j])^2)
      term2= sum((xmat2[,j] - meanmat[2,j])^2)
      svec[j] = sqrt((term1+term2)/(N-G))
      }
   pi1 = N1/N
   pi2 = N2/N

  yvec.predict = NULL

   for (i in 1:nrow(testdata.x)) {
    delta1 = - sum((testdata.x[i,]-meanmat[1,])^2/svec^2) + 2*log(pi1)
    delta2 = - sum((testdata.x[i,]-meanmat[2,])^2/svec^2) + 2*log(pi2)
    yvec.predict[i] = ifelse(delta1<delta2, 1, 0)
   }

    return(yvec.predict)
}
```

B.14 Implementing the NSC classifier to the binary data as in Section 2.5

```
>  fix(Pnsc_class)
function (traindata.x,traindata.y,testdata.x, delta) {
  # delta = the optimal NSC Δ value from thresholding
#########################################################################
#                                                                      ##
# This program implements the NSC classifier on training & test data   ##
#                                                                      ##
#########################################################################
 ## Data properties ##
 xmat = traindata.x
 G =length(unique(traindata.y))
 xmat1 = xmat[traindata.y==0,]
 xmat2 = xmat[traindata.y==1,]
 N1 = nrow(xmat1)
 N2 = nrow(xmat2)
 N = N1+N2                     #number of observations
 p = ncol(xmat)             #number of variables (p)

#########################################################################
#                                                                      ##
# Computing the NSC discriminant score for the 2 classes as in (2.6)   ##
#  and outputting the classification                                   ##
#                                                                      ##
#########################################################################
   meanmat = matrix(0,nrow=2,ncol=p)
   meanmat[1,] = apply(xmat1,2,mean)
   meanmat[2,] = apply(xmat2,2,mean)
   overallmean = apply(xmat,2,mean)

  svec = NULL
  for (j in 1:p) {
   term1= sum((xmat1[,j] - meanmat[1,j])^2)
   term2= sum((xmat2[,j] - meanmat[2,j])^2)
```

```
    svec[j] = sqrt((term1+term2)/(N-G))
   }

   snull = median(svec)       #small positive regularisation parameter
    mvec = NULL
    mvec[1] = sqrt((1/N1)+(1/N))
    mvec[2] = sqrt((1/N2)+(1/N))

    dmat = matrix(0,nrow=2,ncol=p)
    pi1 = N1/N
    pi2 = N2/N

   for (g in 1:2) for (j in 1:p)
        dmat[g,j]= (meanmat[g,j]-overallmean[j])/(mvec[g]*(svec[j]+snull))

     dmat_prime = matrix(0,nrow=2,ncol=p)
     meanmat_prime = matrix(0,nrow=2,ncol=p)
     diff1= matrix(0,nrow=2,ncol=p)

    for (g in 1:2) for (j in 1:p) {
      diff = abs(dmat[g,j])-delta
      diff1[g,j]=ifelse (diff<0,0,diff)
      dmat_prime[g,j] = sign(dmat[g,j])*diff1[g,j]
      meanmat_prime[g,j]= overallmean[j] +
                          mvec[g]*(svec[j]+snull)*dmat_prime[g,j]
    }

## Classifying the test case ##
 yvec.predict = NULL
 for (i in 1:nrow(testdata.x)) {
  delta1= -sum((testdata.x[i,]-meanmat_prime[1,])^2/(svec+snull)^2)
                + 2*log(pi1)
  delta2= -sum((testdata.x[i,]-meanmat_prime[2,])^2/(svec+snull)^2)
              + 2*log(pi2)
  yvec.predict[i] = ifelse(delta1<delta2, 1, 0)
  }

    return(yvec.predict)
}
```

B.15 Implementing the NSC classifier on the multi-class data set

```
> fix(Pnsc_class_mult)
function (traindata.x,traindata.y,testdata.x, delta) {
  # delta = the optimal NSC Δ value from thresholding
##########################################################################
#                                                                       ##
# This program implements NSC on the multi-class training & test data  ##
#                                                                       ##
##########################################################################
 ## Data properties ##
 xmat = traindata.x    #training xmat
 yvec=traindata.y
 G = length(unique(yvec))  #no of classes
 N=nrow(xmat)
 Ng = rep(0,G)
 indg = matrix(rep(0),ncol=G,nrow=N)

 for (g in 1:G) {
  indg[,g] = ifelse(yvec == g,1,0)
  Ng[g] = sum(indg[,g])
 }
   xmat1 = xmat[yvec==1,]
   xmat2 = xmat[yvec==2,]
   xmat3 = xmat[yvec==3,]
   xmat4 = xmat[yvec==4,]

   p = ncol(xmat)            #number of variables (p)

#########################################################################
#                                                                      ##
# Performing NSC & Computing the NSC discriminant score for G>2        ##
#                                                                      ##
#########################################################################

 ## Performing Nearest Shrunken Centroids ##
   meanmat = matrix(0,nrow=G,ncol=p)
   meanmat[1,] = apply(xmat1,2,mean)
   meanmat[2,] = apply(xmat2,2,mean)
   meanmat[3,] = apply(xmat3,2,mean)
   meanmat[4,] = apply(xmat4,2,mean)
   overallmean = apply(xmat,2,mean)

  svec = NULL

  for (j in 1:p) {
   term1= sum((xmat1[,j] - meanmat[1,j])^2)
   term2= sum((xmat2[,j] - meanmat[2,j])^2)
   term3= sum((xmat3[,j] - meanmat[3,j])^2)
   term4= sum((xmat4[,j] - meanmat[4,j])^2)
   svec[j] = sqrt((term1+term2+term3+term4)/(N-G))
  }

   snull = median(svec)

   mvec = NULL
   mvec[1] = sqrt((1/Ng[1])+(1/N))
   mvec[2] = sqrt((1/Ng[2])+(1/N))
   mvec[3] = sqrt((1/Ng[3])+(1/N))
   mvec[4] = sqrt((1/Ng[4])+(1/N))
   dmat = matrix(0,nrow=G,ncol=p)

   pi1 = Ng[1]/N ; pi2 = Ng[2]/N ; pi3 = Ng[3]/N; pi4 = Ng[4]/N

   for (g in 1:G) for (j in 1:p)
      dmat[g,j] = (meanmat[g,j]-overallmean[j])/(mvec[g]*(svec[j]+snull))

   dmat_prime = matrix(0,nrow=G, ncol=p)
   meanmat_prime = matrix(0,nrow=G, ncol=p)
   diff1= matrix(0,nrow=G, ncol=p)

  for (g in 1:G) for (j in 1:p) {
     diff = abs(dmat[g,j])-delta
     diff1[g,j]=ifelse (diff<0,0,diff)
     dmat_prime[g,j] = sign(dmat[g,j])*diff1[g,j]
     meanmat_prime[g,j]=overallmean[j] +
                           mvec[g]*(svec[j]+snull)*dmat_prime[g,j]
   }

 yvec.predict = NULL
 deltamat = matrix(rep(0),ncol=G,nrow=nrow(testdata.x))
  for (i in 1:nrow(testdata.x)) {

 ## NSC discriminant score for the more than two classes as in (2.3) ##
   delta1= -sum((testdata.x[i,]-meanmat_prime[1,])^2/(svec+snull)^2)
                + 2*log(pi1)
   delta2= -sum((testdata.x[i,]-meanmat_prime[2,])^2/(svec+snull)^2)
                + 2*log(pi2)
   delta3= -sum((testdata.x[i,]-meanmat_prime[3,])^2/(svec+snull)^2)
                + 2*log(pi3)
   delta4= -sum((testdata.x[i,]-meanmat_prime[4,])^2/(svec+snull)^2)
               + 2*log(pi4)
   deltamat[i,] = cbind(delta1, delta2, delta3, delta4)

 ## The classification rule ##
   yvec.predict[i]= which.max(deltamat[i,])
  }

  return(yvec.predict)
}
```

B.16 Applying the NSC VS procedure to binary data as in Section 2.5

```
>  fix(Pnsc)
```

```
function (data.xy1, delta) {
 # data.xy1 refers to data containing both the y & x variables
 # delta = the optimal NSC Δ value from thresholding

#########################################################################
#                                                                     ##
# This program applies the NSC selection procedure to binary data     ##
#                                                                     ##
#########################################################################

## Data Properties ##
 col1 = ncol(data.xy1)   #column with y variables
 xmat= as.matrix(data.xy1[,-col1],nrow=ndata,ncol=nvars)   #training xmat
 yvec=data.xy1[,col1]
 G =length(unique(yvec))  #No. of population groups

 xmat1 = xmat[yvec==0,]
 xmat2 = xmat[yvec==1,]

 N1 = nrow(xmat1)
 N2 = nrow(xmat2)
 N = N1+N2                    #number of observations
 p = ncol(xmat)              #number of variables (p)

## Method 1: Performing Nearest Shrunken Centroids ##
  meanmat = matrix(0,nrow=2,ncol=p)
  meanmat[1,] = apply(xmat1,2,mean)
  meanmat[2,] = apply(xmat2,2,mean)
  overallmean = apply(xmat,2,mean)

  svec = NULL

  for (j in 1:p) {
   term1= sum((xmat1[,j] - meanmat[1,j])^2)
   term2= sum((xmat2[,j] - meanmat[2,j])^2)
```

```
    svec[j] = sqrt((term1+term2)/(N-G))
   }

    snull = median(svec)
    mvec = NULL
    mvec[1] = sqrt((1/N1)+(1/N))
    mvec[2] = sqrt((1/N2)+(1/N))

    dmat = matrix(0,nrow=2,ncol=p)

     for (g in 1:2) for (j in 1:p)
        dmat[g,j]= (meanmat[g,j]-overallmean[j])/(mvec[g]*(svec[j]+snull))

    dmat_prime = matrix(0,nrow=2,ncol=p)
    meanmat_prime = matrix(0,nrow=2,ncol=p)
    diff1= matrix(0,nrow=2,ncol=p)

    for (g in 1:2) for (j in 1:p) {
       diff = abs(dmat[g,j])-delta
       diff1[g,j]=ifelse (diff<0,0,diff)
       dmat_prime[g,j] = sign(dmat[g,j])*diff1[g,j]
       meanmat_prime[g,j] = overallmean[j] +
                                    mvec[g]*(svec[j]+snull)*dmat_prime[g,j]
    }
##########################################################################
#                                                                       ##
# Outputting the NSC selected variables                                 ##
#                                                                       ##
##########################################################################
output= list("meanmat_prime"= meanmat_prime,"svec"=svec,"snull"=snull,
    "N1"=N1,"N2"=N2,"Dmat"=dmat,"Dmat_prime"=dmat_prime,"nsc.omit"=diff1)

return(output)
}
```

B.17 Applying the NSC VS procedure to multi-class data set

```
> fix(Pnsc_mult)
function (data.xy1, delta) {
 # data.xy1 refers to data containing both the y & x variables
 # delta = the optimal NSC Δ value from thresholding

##########################################################################
#                                                                       ##
# This program applies the NSC VS procedure to multiclass data          ##
#                                                                       ##
##########################################################################
## Data Properties ##
 col1 = ncol(data.xy1)                     #column with y variables
 xmat = as.matrix(data.xy1[,-col1],nrow=ndata,ncol=nvars)   #training xmat
 yvec=data.xy1[,col1]
 G = length(unique(yvec))  #no of classes
 N=nrow(xmat)
 Ng = rep(0,G)
 indg = matrix(rep(0),ncol=G,nrow=N)
 for (g in 1:G) {
  indg[,g] = ifelse(yvec == g,1,0)
  Ng[g] = sum(indg[,g])
 }
```

```
    xmat1 = xmat[yvec==1,]
    xmat2 = xmat[yvec==2,]
    xmat3 = xmat[yvec==3,]
    xmat4 = xmat[yvec==4,]
    p = ncol(xmat)            #number of variables (p)

  ## Performing Nearest Shrunken Centroids ##
    meanmat = matrix(0,nrow=G,ncol=p)
    meanmat[1,] = apply(xmat1,2,mean)
    meanmat[2,] = apply(xmat2,2,mean)
    meanmat[3,] = apply(xmat3,2,mean)
    meanmat[4,] = apply(xmat4,2,mean)
    overallmean = apply(xmat,2,mean)

   svec = NULL

   for (j in 1:p) {
    term1= sum((xmat1[,j] - meanmat[1,j])^2)
    term2= sum((xmat2[,j] - meanmat[2,j])^2)
    term3= sum((xmat3[,j] - meanmat[3,j])^2)
    term4= sum((xmat4[,j] - meanmat[4,j])^2)
    svec[j] = sqrt((term1+term2+term3+term4)/(N-G))
   }
    snull = median(svec)
    mvec = NULL
    mvec[1] = sqrt((1/Ng[1])+(1/N))
    mvec[2] = sqrt((1/Ng[2])+(1/N))
    mvec[3] = sqrt((1/Ng[3])+(1/N))
    mvec[4] = sqrt((1/Ng[4])+(1/N))

    dmat = matrix(0,nrow=G,ncol=p)
     for (g in 1:G) for (j in 1:p)
        dmat[g,j]= (meanmat[g,j]-overallmean[j])/(mvec[g]*(svec[j]+snull))

    dmat_prime = matrix(0,nrow=G, ncol=p)
    meanmat_prime = matrix(0,nrow=G, ncol=p)
    diff1= matrix(0,nrow=G, ncol=p)

    for (g in 1:G) for (j in 1:p) {
       diff = abs(dmat[g,j])-delta
       diff1[g,j]=ifelse (diff<0,0,diff)
       dmat_prime[g,j] = sign(dmat[g,j])*diff1[g,j]
       meanmat_prime[g,j] = overallmean[j] +
                            mvec[g]*(svec[j]+snull)*dmat_prime[g,j]
    }

#########################################################################
#                                                                      ##
# Outputting the NSC selected variables                                ##
#                                                                      ##
#########################################################################
output= list("meanmat_prime"= meanmat_prime,"svec"=svec,"snull"=snull,
             "Dmat"=dmat,"Dmat_prime"= dmat_prime,"nsc.omit"=diff1)

return(output)
 }
```

B.18 Applying the SVM classifier to the binary data as in Section 2.5

```
>  fix(Psvm)
function (traindata.x,traindata.y,testdata.x, Cpar) {
 #Cpar = cost parameter value in RBF SVM C ϵ {1,1000,10000}
########################################################################
#                                                                     ##
# This program determines in F values for the RBF SVM classifier      ##
#                                                                     ##
########################################################################

## Standardise the training and test data ##
    meanx = apply(traindata.x, 2, mean)
    sdx = apply(traindata.x, 2, sd)
    traindata.x = scale(traindata.x, center = meanx, scale = sdx)
    testdata.x = scale(testdata.x, center = meanx, scale = sdx)

########################################################################
#                                                                     ##
# Finding the optimal gamma value based on the training data          ##
#                                                                     ##
########################################################################
  findgam = sigest(as.matrix(traindata.x), frac=1, scaled=TRUE)
  opt.gam = findgam[2]
  rbf  = rbfdot(sigma=opt.gam)

########################################################################
#                                                                     ##
# Determining the F values from the RBF SVM based on optimal gamma val ##
#                                                                     ##
########################################################################
 svmfit= ksvm(x=traindata.x, y=as.factor(traindata.y), type="C-svc",
       kernel="rbfdot",kpar=list(sigma=opt.gam),C=Cpar, prob.model=FALSE)
  coefsvm = as.matrix(unlist(coef(svmfit)))
  bcoef = unlist(b(svmfit))
  index = unlist(SVindex(svmfit))
  coefval = rep(0, nrow(traindata.x))
  coefval[index] = coefsvm
  gmat = kernelMatrix(rbf, testdata.x, traindata.x)
  q3 = matrix(t(as.matrix(coefval)) %*% t(gmat), ncol=1)
  fvalues = q3-bcoef[1]     #real F value
   # note that sign(fvalues) = -1 or 1

## Outputting the F values ##
   return(fvalues)
}
```

B.19 Performing correlation thresholding VS procedure as in Section 3.3.1

```
>  fix(Pcor)
function(data.xy, threshold) {
 #threshold = correlation coefficient threshold value
########################################################################
#                                                                     ##
# This program performs correlation thresholding on data.xy           ##
#   in order to determine the original input variables to retain      ##
#                                                                     ##
########################################################################
## Obtain matrix of input values & binary response vector from data.xy ##
 ncol = ncol(data.xy)
 p = ncol-1
 N = nrow(data.xy)
 xmat = as.matrix(data.xy[,-ncol])
 yvec = data.xy[,ncol]

##########################################################################
#                                                                       ##
# Compute the correlation coefficients & identify variables to retained #
#                                                                       ##
##########################################################################
 corrv= cor(xmat, yvec)
 retain.ind = which(abs(corrv) > threshold)

 output = list("Correlations" = corrv, "Indices" = retain.ind)
 return(output)
}
```

B.20 Performing the two-sample *t*-test VS procedure as in Section 3.3.2

```
> fix(Pttest)
```

```
function(data.xy, threshold) {
  #threshold = specified p-value thresholding
##########################################################################
#                                                                       ##
# This program performs pairwise t-tests in order to determine          ##
#    which original input variables to retain                           ##
# input variable is retained if calc. t-test p-value<specified thresh   ##
#                                                                       ##
##########################################################################

## Obtain matrix of input values & binary response vector from data.xy ##
 ncol = ncol(data.xy)
 p = ncol-1
 N = nrow(data.xy)
 xmat = as.matrix(data.xy[,-ncol])
 yvec = data.xy[,ncol]
 ind0 = which(yvec == 0)
 ind1 = which(yvec == 1)
 N0 = length(ind0)
 N1 = length(ind1)
 xmat.y0 = xmat[ind0,]
 xmat.y1 = xmat[ind1,]

##########################################################################
#                                                                       ##
# Performing the t-test & identifying which variables to select       ##
#                                                                       ##
##########################################################################
 mean.x0 = apply(xmat.y0, 2, mean)
 mean.x1 = apply(xmat.y1, 2, mean)

 SS = function(w) {
    mw = mean(w)
    sum((w - mw)^2)
   }
 ss0 = apply(xmat.y0, 2, SS)
 ss1 = apply(xmat.y1, 2, SS)
```

```
  se.pooled = sqrt((ss0+ss1)/(N0+N1-2)) * sqrt(1/N0+1/N1)

  tstat = (mean.x1- mean.x0)/se.pooled
  p.values = 1 - pt(abs(tstat), df = N0+N1-2)
  retain.ind = which(p.values < threshold)
  output = list("t.statistics" = tstat, "Indices" = retain.ind)
  return(output)
 }
```

B.21 Using LOOCV to determine optimal threshold values for colon & leukemia data sets reported in Section 5.6

```
> fix(Pthreshold)
```

```
function(data.xy, Kvec, Cpar, numsplits, trfrac,t.testthr) {
    # Kvec= vector containing 3 possible values for the K in KNN
    # Cpar= vector containing 3 possible values for C parameter in SVM

##########################################################################
#                                                                       ##
# This program determines the threshold values to be used in the:      ##
#    a) Two-sample t-test                                               ##
#    b) Correlation thresholding  &                                     ##
#    c) NSC VS procedure                                                ##
#                                                                       ##
##########################################################################

## Obtain matrix of input values & binary response vector from data.xy ##
  ncol = ncol(data.xy)
  p = ncol-1
  N = nrow(data.xy)
  data.x = as.matrix(data.xy[,-ncol], nrow=N, ncol=p)
  yvec = data.xy[,ncol]
  data.y = yvec

 tknn.outA= tknn.outB = tknn.outC = matrix(rep(0),ncol=1,nrow=numsplits)
 corknn.outA=corknn.outB=corknn.outC=matrix(rep(0),ncol=1,nrow=numsplits)
 nscknn.outA=nscknn.outB=nscknn.outC=matrix(rep(0),ncol=1,nrow=numsplits)
 ttest.outA=ttest.outB=ttest.outC = matrix(rep(0),ncol=1,nrow=numsplits)
 cor.outA = cor.outB = cor.outC = matrix(rep(0),ncol=1,nrow=numsplits)
 nsc.outA = nsc.outB = nsc.outC = matrix(rep(0),ncol=1,nrow=numsplits)
 nsc.out = matrix(rep(0),ncol=1,nrow=numsplits)

##########################################################################
#                                                                       ##
# Obtaining the candidate corr. & NSC threshold values frm data.xy     ##
#                                                                       ##
##########################################################################
  ## Obtaining the 5 candidate correlation threshold values ##
   corvec = abs(cor(data.y,data.x))
   cor.thr  = seq(from=0.1, to=0.8*max(corvec), length=5)

  ## Obtain 15 candidate NSC threshold values from pamr package ##
    pam.x = t(data.x)
    pam.y = as.factor(data.y)
    pam.data = list(x=pam.x, y=pam.y)
    pamr.threshold = pamr.train(pam.data,n.threshold=30)$threshold
    pamr.thr = pamr.threshold[8:22]
```

```
##########################################################################
#                                                                        ##
# Performing LOOCV to find the optimal t-test,correlation & NSC values ##
#                                                                        ##
##########################################################################

## Splitting the data into training & test for LOOCV ##
  ind0 = which(yvec == 0)
  ind1 = which(yvec == 1)
  N0 = length(ind0)
  N1 = length(ind1)
  trainsize0 = floor(trfrac*N0)
  trainsize1 = floor(trfrac*N1)
  trainsize = trainsize0 + trainsize1
  testsize0 = N0 - trainsize0
  testsize1 = N1 - trainsize1
  testsize = testsize0 + testsize1

# while loop for repeatedly splitting data into training & test parts
  nm = 0

  while (nm<numsplits)  { #A
    ok = TRUE  # Intially the split is fine

    trainind0 = sample (ind0, trainsize0, replace=F)
    trainind1 = sample (ind1, trainsize1, replace=F)
    trainind = c(trainind0,trainind1)
    traindata.x = data.x[trainind,]
    traindata.y = data.y[trainind]
    traindata.xy = cbind(traindata.x, traindata.y)
    testdata.x = data.x[-trainind,]
    testdata.y = data.y[-trainind]

     # creating 3-dimensional arrays
     nscAr = array(0, c(nrow(traindata.x), p, length(pamr.thr)))
     ttestAr = array(0, c(nrow(traindata.x), p, length(t.testthr)))
     corAr = array(0, c(nrow(traindata.x), p, length(cor.thr)))

   # Note that we are currently inside the loop splitting the data into
training and test parts
       for (i in 1:nrow(traindata.x)) {            #B: LOOCV loop
          xtrain = traindata.x[-i,]
          ytrain = traindata.y[-i]
          xytrain = cbind(xtrain, ytrain)
          xtest = matrix(traindata.x[i,],nrow=1,ncol=p)
          ytest = traindata.y[i]


          ## Performing NSC thresholding ##
              if (ok == TRUE) {  #BB
               for (d in 1:length(pamr.thr)) {      #C
                  threshold = pamr.thr[d]
                  nsc = Pnsc(xytrain,threshold)
                  xindnsc = which(apply(nsc$nsc.omit,2,sum)>0)

                  if (length(xindnsc)==0) ok=FALSE
                  if (length(xindnsc)>0)   nscAr[i, xindnsc, d] =1
                } #C
              } #BB
          ## Performing T-test thresholding ##
              if (ok == TRUE) {   #D
               for (j in 1:length(t.testthr)) { #E
                  threshold = t.testthr[j]
                  tout = Pttest(xytrain, threshold)
                  xindt = tout$Indices

                  if (length(xindt)==0) ok=FALSE
                  if (length(xindt)>0)   ttestAr[i, xindt, j] =1
                } #E
               } # D


          ## Performing Correlation thresholding ##
              if (ok == TRUE) {   #F
               for (j in 1:length(cor.thr)) { #G
                  threshold = cor.thr[j]
                  corout = Pcor(xytrain, threshold)
                  xindcor = corout$Indices

                  if (length(xindcor)==0) ok=FALSE
                  if (length(xindcor)>0)   corAr[i, xindcor, j] =1
                   }  #G
                } # F

         } #B (closing LOOCV)

        # Now LOOCV errors can be computed if ok still =TRUE
        if(ok==TRUE) { # H

        knn.tcvA= knn.tcvB= knn.tcvC= matrix(0, ncol = length(t.testthr),
nrow = nrow(traindata.x))
        knn.corcvA =knn.corcvB=knn.corcvC= matrix(0, ncol =
length(cor.thr), nrow = nrow(traindata.x))
        knn.nsccvA =knn.nsccvB=knn.nsccvC= matrix(0, ncol = 15, nrow =
nrow(traindata.x))

        error.tcvA = error.tcvB = error.tcvC = matrix(0, ncol =
length(t.testthr), nrow = nrow(traindata.x))
        error.corcvA = error.corcvB = error.corcvC = matrix(0, ncol =
length(cor.thr), nrow = nrow(traindata.x))
        error.nsccvA = error.nsccvB = error.nsccvC = matrix(0, ncol = 15,
nrow = nrow(traindata.x))

        error.NSCcv = matrix(0, ncol = 15, nrow = nrow(traindata.x))

      for (i in 1:nrow(traindata.x)) {            # I
         xtrain = traindata.x[-i,]
         ytrain = traindata.y[-i]
         xytrain = cbind(xtrain, ytrain)
         xtest = matrix(traindata.x[i,],nrow=1,ncol=p)
         ytest = traindata.y[i]


     ## Compute an error for every NSC threshold in pamr ##
         for (d in 1:length(pamr.thr)) {      # J
             threshold = pamr.thr[d]
             xindnsc = which(nscAr[i, ,d]==1)
             x.reducednsc = matrix(xtrain[,xindnsc],nrow=nrow(xtrain),
                            ncol=length(xindnsc))
```

```
            ## KNN ##
               out = knn(train = x.reducednsc, cl = as.factor(ytrain),
                          test = traindata.x[i,xindnsc], k=Kvec[1])
                knn.nsccvA[i, d] = (as.numeric(out)-1) != traindata.y[i]

                out = knn(train = x.reducednsc, cl = as.factor(ytrain),
                            test = traindata.x[i,xindnsc], k=Kvec[2])
                knn.nsccvB[i, d] = (as.numeric(out)-1) != traindata.y[i]

                out = knn(train = x.reducednsc, cl = as.factor(ytrain),
                           test = traindata.x[i,xindnsc], k=Kvec[3])
                knn.nsccvC[i, d] = (as.numeric(out)-1) != traindata.y[i]

              ## SVM ##
               out = Psvm(x.reducednsc,ytrain,matrix(xtest[,xindnsc],
                            nrow=1,ncol=length(xindnsc)),Cpar[1])
                error.nsccvA[i, d] = as.numeric((sign(out)+1)/2 != ytest)

                out = Psvm(x.reducednsc,ytrain, matrix(xtest[,xindnsc],
                             nrow=1,ncol=length(xindnsc)),Cpar[2])
                error.nsccvB[i, d] = as.numeric((sign(out)+1)/2 != ytest)

                out = Psvm(x.reducednsc,ytrain,matrix(xtest[,xindnsc],
                             nrow=1,ncol=length(xindnsc)),Cpar[3])
                error.nsccvC[i, d] = as.numeric((sign(out)+1)/2 != ytest)

              ## NSC ##
                out = Pnsc_class(xtrain, ytrain, xtest, delta=threshold)
                error.NSCcv[i, d] = sum(as.numeric(out!=  traindata.y[i]))

             } # J

        ## Compute an error for every threshold in t-test ##
            for (j in 1:length(t.testthr)) {     # K
                threshold = t.testthr[j]
                xindt = which(ttestAr[i, ,j]==1)
                x.reducedt =
matrix(xtrain[,xindt],nrow=nrow(xtrain),ncol=length(xindt))

              ## KNN ##
                out = knn(train = x.reducedt, test = traindata.x[i,xindt],
cl = as.factor(ytrain), k=Kvec[1])
                knn.tcvA[i, j] = (as.numeric(out)-1) != traindata.y[i]

                out = knn(train = x.reducedt, test = traindata.x[i,xindt],
cl = as.factor(ytrain), k=Kvec[2])
                knn.tcvB[i, j] = (as.numeric(out)-1) != traindata.y[i]

                out = knn(train = x.reducedt, test = traindata.x[i,xindt],
cl = as.factor(ytrain), k=Kvec[3])
                knn.tcvC[i, j] = (as.numeric(out)-1) != traindata.y[i]


              ## SVM ##
                out =
Psvm(x.reducedt,ytrain,matrix(xtest[,xindt],nrow=1,ncol=length(xindt)),Cp
ar[1])
                error.tcvA[i, j] = as.numeric((sign(out)+1)/2 != ytest)
#sign(fvalues) = -1 or 1
```

```
               out =
Psvm(x.reducedt,ytrain,matrix(xtest[,xindt],nrow=1,ncol=length(xindt)),Cp
ar[2])
               error.tcvB[i, j] = as.numeric((sign(out)+1)/2 != ytest)

               out =
Psvm(x.reducedt,ytrain,matrix(xtest[,xindt],nrow=1,ncol=length(xindt)),Cp
ar[3])
               error.tcvC[i, j] = as.numeric((sign(out)+1)/2 != ytest)
             } # K

        ## Compute an error for every threshold in corr ##
            for (j in 1:length(cor.thr)) {     # L
                threshold = cor.thr[j]
                xindcor = which(corAr[i, ,j]==1)
                x.reducedcor =
matrix(xtrain[,xindcor],nrow=nrow(xtrain),ncol=length(xindcor))

             ## KNN ##
               out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[1])
               knn.corcvA[i, j] = (as.numeric(out)-1) != traindata.y[i]

               out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[2])
               knn.corcvB[i, j] = (as.numeric(out)-1) != traindata.y[i]

               out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[3])
               knn.corcvC[i, j] = (as.numeric(out)-1) != traindata.y[i]


             ## SVM ##
               out = Psvm(x.reducedcor,ytrain,matrix(xtest[,xindcor],
                          nrow=1,ncol=length(xindcor)),Cpar[1])
               error.corcvA[i, j] = as.numeric((sign(out)+1)/2 != ytest)

               out = Psvm(x.reducedcor,ytrain, matrix(xtest[,xindcor],
                          nrow=1,ncol=length(xindcor)),Cpar[2])
               error.corcvB[i, j] = as.numeric((sign(out)+1)/2 != ytest)

               out = Psvm(x.reducedcor,ytrain, matrix(xtest[,xindcor],
                     nrow=1,ncol=length(xindcor)),Cpar[3])
               error.corcvC[i, j] =as.numeric((sign(out)+1)/2 != ytest)
            } # L


       } #I (end of error computations)


          nm = nm +1

      ## t-test optimal p-value ##

             ## KNN ##
              tot.error.cv.t = apply(knn.tcvA, 2, sum)
              optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
              tknn.outA[nm] = optimal.tthreshold

             tot.error.cv.t = apply(knn.tcvB, 2, sum)
             optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
```

```
              tknn.outB[nm] = optimal.tthreshold

               tot.error.cv.t = apply(knn.tcvC, 2, sum)
               optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
               tknn.outC[nm] = optimal.tthreshold

              ## SVM ##
               tot.error.cv.t = apply(error.tcvA, 2, sum)
               optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
               ttest.outA[nm] = optimal.tthreshold

               tot.error.cv.t = apply(error.tcvB, 2, sum)
               optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
               ttest.outB[nm] = optimal.tthreshold

               tot.error.cv.t = apply(error.tcvC, 2, sum)
               optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
               ttest.outC[nm] = optimal.tthreshold

        ## optimal correlation threshold value##
              ## KNN ##
               tot.error.cv.cor = apply(knn.corcvA, 2, sum)
               optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
               corknn.outA[nm] = optimal.corthreshold

               tot.error.cv.cor = apply(knn.corcvB, 2, sum)
               optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
               corknn.outB[nm] = optimal.corthreshold

               tot.error.cv.cor = apply(knn.corcvC, 2, sum)
               optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
               corknn.outC[nm] = optimal.corthreshold


              ## SVM ##
               tot.error.cv.cor = apply(error.corcvA, 2, sum)
               optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
               cor.outA[nm] = optimal.corthreshold

               tot.error.cv.cor = apply(error.corcvB, 2, sum)
               optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
               cor.outB[nm] = optimal.corthreshold

               tot.error.cv.cor = apply(error.corcvC, 2, sum)
               optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
               cor.outC[nm] = optimal.corthreshold

        ## NSC optimal threshold ##

              ## KNN ##
               tot.error.cv.nsc = apply(knn.nsccvA, 2, sum)
                  min.error=min(tot.error.cv.nsc)
               nscknn.outA[nm]=
pamr.thr[max(which(tot.error.cv.nsc==min.error))]

               tot.error.cv.nsc = apply(knn.nsccvB, 2, sum)
                  min.error=min(tot.error.cv.nsc)
               nscknn.outB[nm] =
pamr.thr[max(which(tot.error.cv.nsc==min.error))]

               tot.error.cv.nsc = apply(knn.nsccvC, 2, sum)
```

```
                   min.error=min(tot.error.cv.nsc)
               nscknn.outC[nm] = 
pamr.thr[max(which(tot.error.cv.nsc==min.error))]

              ## SVM ##
               tot.error.cv.nsc = apply(error.nsccvA, 2, sum)    #cv for 
single split
               min.error=min(tot.error.cv.nsc)

nsc.outA[nm]=pamr.thr[max(which(tot.error.cv.nsc==min.error))]

               tot.error.cv.nsc = apply(error.nsccvB, 2, sum)
               min.error=min(tot.error.cv.nsc)

nsc.outB[nm]=pamr.thr[max(which(tot.error.cv.nsc==min.error))]

               tot.error.cv.nsc = apply(error.nsccvC, 2, sum)
               min.error=min(tot.error.cv.nsc)

nsc.outC[nm]=pamr.thr[max(which(tot.error.cv.nsc==min.error))]

              ## NSC ##
               tot.error.cv.nsc = apply(error.NSCcv, 2, sum)
               min.error=min(tot.error.cv.nsc)
               nsc.out[nm] = 
pamr.thr[max(which(tot.error.cv.nsc==min.error))]

         } #H
        print(nm)
     }  #A (end of nm CV)

##########################################################################
#                                                                       ##
# Returning the selection frequencies of the candidate thresh. values   ##
#                                                                       ##
##########################################################################
 ## KNN classifier's Selection Frequencies ##
    tA = table(tknn.outA)
    tB = table(tknn.outB)
    tC = table(tknn.outC)

    cA = table(corknn.outA)
    cB = table(corknn.outB)
    cC = table(corknn.outC)

    nA = table(nscknn.outA)
    nB = table(nscknn.outB)
    nC = table(nscknn.outC)

 ## SVM classifier's selection Frequencies ##
     ttestA.freq = table(ttest.outA)
     ttestB.freq = table(ttest.outB)
     ttestC.freq = table(ttest.outC)

     corA.freq = table(cor.outA)
     corB.freq = table(cor.outB)
     corC.freq = table(cor.outC)

     nscA.freq = table(nsc.outA)
     nscB.freq = table(nsc.outB)
     nscC.freq = table(nsc.outC)
```

```
  ## NSC classifier's selection frequencies ##
     NSC.freq = table(nsc.out)

## Outputting the results ##
 output= list("KNN ttestA freq"=tA, "KNN ttest B freq"=tB,
         "KNN ttestC freq"= tC, "KNN corA freq"=cA,"KNN corB freq"=cB,
         "KNN corC freq"=cC, "15 possible NSC thresholds"=pamr.thr,
         "KNN nscA freq"=nA,"KNN nscB freq"=nB,"KNN nscC freq"=nC,

         "SVM ttestA freq"=ttestA.freq, "SVM ttest B freq"=ttestB.freq,
         "SVM ttestC freq"= ttestC.freq,"SVM corA freq"=corA.freq,
         "SVM corB freq"=corB.freq, "SVM corC freq"= corC.freq,
         "15 possible NSC thresholds"=pamr.thr,
         "SVM nscA freq" = nscA.freq, "SVM nscB freq" = nscB.freq,
         "SVM nscC freq" = nscC.freq,"NSC classifier freq" = NSC.freq)

  return(output)
}
```

B.22 Using LOOCV to determine optimal threshold values for Sim1, Sim2 & Sim3 data reported in Section 5.6

```
> fix(Pthreshold_sim)
```

```
function(data.xy, Kvec, Cpar, numsplits, trfrac,t.testthr) {
  # Kvec = vector containing 3 possible values for the K in KNN
  # Cpar = vector containing 3 possible values for the C parameter in SVM

#########################################################################
#                                                                     ##
# This program determines the threshold values to be used in the:     ##
#    a) Two-sample t-test                                             ##
#    b) Correlation thresholding  &                                   ##
#    c) NSC VS procedure                                              ##
#                                                                     ##
#########################################################################

## Obtain matrix of input values & binary response vector from data.xy ##

  ncol = ncol(data.xy)
  p = ncol-1
  N = nrow(data.xy)
  data.x = as.matrix(data.xy[,-ncol], nrow=N, ncol=p)
  yvec = data.xy[,ncol]
  data.y = yvec

 tknn.outA = tknn.outB= tknn.outC = matrix(rep(0),ncol=1,nrow=numsplits)
 corknn.outA=corknn.outB=corknn.outC=matrix(rep(0),ncol=1,nrow=numsplits)
 ttest.outA = ttest.outB=ttest.outC= matrix(rep(0),ncol=1,nrow=numsplits)
 cor.outA = cor.outB = cor.outC = matrix(rep(0),ncol=1,nrow=numsplits)
 nsc.out = matrix(rep(0),ncol=1,nrow=numsplits)

#########################################################################
#                                                                     ##
# Obtaining the candidate corr. & NSC threshold values frm data.xy    ##
#                                                                     ##
#########################################################################
```

```
 ## Obtaining the 5 candidate correlation threshold values ##
   corvec = abs(cor(data.y,data.x))
   cor.thr  = seq(from=0.1, to=0.8*max(corvec), length=5)

  ## Obtain 15 candidate NSC threshold values from pamr package ##
    pam.x = t(data.x)
    pam.y = as.factor(data.y)
    pam.data = list(x=pam.x, y=pam.y)
    pamr.threshold = pamr.train(pam.data,n.threshold=30)$threshold
    pamr.thr = pamr.threshold[8:22]

#########################################################################
#                                                                      ##
# Performing LOOCV to find the optimal t-test,correlation & NSC values ##
#                                                                      ##
#########################################################################

## Splitting the data into training & test for LOOCV ##
    ind0 = which(yvec == 0)
    ind1 = which(yvec == 1)
    N0 = length(ind0)
    N1 = length(ind1)

    trainsize0 = floor(trfrac*N0)
    trainsize1 = floor(trfrac*N1)
    trainsize = trainsize0 + trainsize1
    testsize0 = N0 - trainsize0
    testsize1 = N1 - trainsize1
    testsize = testsize0 + testsize1

 # The while loop for repeatedly splitting the data into training and
test parts starts here
    nm = 0
     while (nm<numsplits)  { #A

     ok = TRUE  # Intially the split is fine

        trainind0 = sample (ind0, trainsize0, replace=F)
        trainind1 = sample (ind1, trainsize1, replace=F)
        trainind = c(trainind0,trainind1)
        traindata.x = data.x[trainind,]
        traindata.y = data.y[trainind]
        traindata.xy = cbind(traindata.x, traindata.y)
        testdata.x = data.x[-trainind,]
        testdata.y = data.y[-trainind]

     # creating 3-dimensional arrays
     nscAr = array(0, c(nrow(traindata.x), p, length(pamr.thr)))
     ttestAr = array(0, c(nrow(traindata.x), p, length(t.testthr)))
     corAr = array(0, c(nrow(traindata.x), p, length(cor.thr)))

  # Note that we are currently inside the loop splitting the data into
training and test parts
     for (i in 1:nrow(traindata.x)) {           #B: LOOCV loop
         xtrain = traindata.x[-i,]
         ytrain = traindata.y[-i]
         xytrain = cbind(xtrain, ytrain)
         xtest = matrix(traindata.x[i,],nrow=1,ncol=p)
         ytest = traindata.y[i]
```

```
       ## NSC thresholding ##
          if (ok == TRUE) {  #BB
             for(d in 1:length(pamr.thr)) {     #C
                threshold = pamr.thr[d]
                nsc = Pnsc(xytrain,threshold)
                xindnsc = which(apply(nsc$nsc.omit,2,sum)>0)

                if (length(xindnsc)==0) ok=FALSE
                if (length(xindnsc)>0)   nscAr[i, xindnsc, d] =1
              } #C
           } #BB


        ## T-test thresholding ##
          if (ok == TRUE) {  #D
              for(j in 1:length(t.testthr)) { #E
                 threshold = t.testthr[j]
                 tout = Pttest(xytrain, threshold)
                 xindt = tout$Indices

                 if (length(xindt)==0) ok=FALSE
                 if (length(xindt)>0)   ttestAr[i, xindt, j] =1
                } #E
            } # D


        ## Correlation thresholding ##
          if (ok == TRUE) {  #F
              for (j in 1:length(cor.thr)) { #G
                 threshold = cor.thr[j]
                 corout = Pcor(xytrain, threshold)
                 xindcor = corout$Indices

                 if (length(xindcor)==0) ok=FALSE
                 if (length(xindcor)>0)   corAr[i, xindcor, j] =1
               }  #G
           } # F

       } #B (closing LOOCV)

    # Now LOOCV errors can be computed if ok still =TRUE
      if(ok==TRUE) { # H

        knn.tcvA= knn.tcvB= knn.tcvC= matrix(0, ncol = length(t.testthr),
nrow = nrow(traindata.x))
        knn.corcvA =knn.corcvB=knn.corcvC= matrix(0, ncol =
length(cor.thr), nrow = nrow(traindata.x))

        error.tcvA = error.tcvB = error.tcvC = matrix(0, ncol =
length(t.testthr), nrow = nrow(traindata.x))
        error.corcvA = error.corcvB = error.corcvC = matrix(0, ncol =
length(cor.thr), nrow = nrow(traindata.x))

        error.NSCcv = matrix(0, ncol = 15, nrow = nrow(traindata.x))

     for (i in 1:nrow(traindata.x)) {           # I
             xtrain = traindata.x[-i,]
             ytrain = traindata.y[-i]
             xytrain = cbind(xtrain, ytrain)
             xtest = matrix(traindata.x[i,],nrow=1,ncol=p)
             ytest = traindata.y[i]
```

```
########################################################################
#                                                                    ##
# Computing the error for every NSC delta candidate threshold value   ##
#                                                                    ##
########################################################################
      for (d in 1:length(pamr.thr)) {     # J
           threshold = pamr.thr[d]
           xindnsc = which(nscAr[i, ,d]==1)
           x.reducednsc =
matrix(xtrain[,xindnsc],nrow=nrow(xtrain),ncol=length(xindnsc))

             ## NSC ##
              out = Pnsc_class(xtrain, ytrain, xtest, delta=threshold)
              error.NSCcv[i, d] = sum(as.numeric(out!=  traindata.y[i]))
         } # J

########################################################################
#                                                                    ##
# Computing the error for every t-test candidate threshold p-value    ##
#                                                                    ##
########################################################################
      for (j in 1:length(t.testthr)) {     # K
        threshold = t.testthr[j]
        xindt = which(ttestAr[i, ,j]==1)
        x.reducedt=matrix(xtrain[,xindt],nrow=nrow(xtrain),
              ncol=length(xindt))

             ## KNN ##
              out = knn(train = x.reducedt, test = traindata.x[i,xindt],
cl = as.factor(ytrain), k=Kvec[1])
              knn.tcvA[i, j] = (as.numeric(out)-1) != traindata.y[i]

              out = knn(train = x.reducedt, test = traindata.x[i,xindt],
cl = as.factor(ytrain), k=Kvec[2])
              knn.tcvB[i, j] = (as.numeric(out)-1) != traindata.y[i]

              out = knn(train = x.reducedt, test = traindata.x[i,xindt],
cl = as.factor(ytrain), k=Kvec[3])
              knn.tcvC[i, j] = (as.numeric(out)-1) != traindata.y[i]

             ## SVM ##
              out = Psvm(x.reducedt,ytrain,matrix(xtest[,xindt],nrow=1,
                         ncol=length(xindt)),Cpar[1])
              error.tcvA[i, j] = as.numeric((sign(out)+1)/2 != ytest)

              out =  Psvm(x.reducedt,ytrain,matrix(xtest[,xindt],nrow=1,
                          ncol=length(xindt)),Cpar[2])
              error.tcvB[i, j] = as.numeric((sign(out)+1)/2 != ytest)

              out =  Psvm(x.reducedt,ytrain,matrix(xtest[,xindt],
                         nrow=1,ncol=length(xindt)),Cpar[3])
              error.tcvC[i, j] = as.numeric((sign(out)+1)/2 != ytest)
         } # K

########################################################################
#                                                                    ##
# Computing the error for every correlation threshold candidate value  ##
#                                                                    ##
########################################################################

         for (j in 1:length(cor.thr)) {     # L
```

```
               threshold = cor.thr[j]
               xindcor = which(corAr[i, ,j]==1)
               x.reducedcor =
matrix(xtrain[,xindcor],nrow=nrow(xtrain),ncol=length(xindcor))

             ## KNN ##
              out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[1])
              knn.corcvA[i, j] = (as.numeric(out)-1) != traindata.y[i]

              out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[2])
              knn.corcvB[i, j] = (as.numeric(out)-1) != traindata.y[i]

              out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[3])
              knn.corcvC[i, j] = (as.numeric(out)-1) != traindata.y[i]

             ## SVM ##
              out = Psvm(x.reducedcor,ytrain,matrix(xtest[,xindcor],
                         nrow=1,ncol=length(xindcor)),Cpar[1])
              error.corcvA[i, j] = as.numeric((sign(out)+1)/2 != ytest)

              out = Psvm(x.reducedcor,ytrain,
matrix(xtest[,xindcor],nrow=1,ncol=length(xindcor)),Cpar[2])
              error.corcvB[i, j] = as.numeric((sign(out)+1)/2 != ytest)

              out = Psvm(x.reducedcor,ytrain,
matrix(xtest[,xindcor],nrow=1,ncol=length(xindcor)),Cpar[3])
              error.corcvC[i, j] =as.numeric((sign(out)+1)/2 != ytest)

          } # L

       } #I (end of error computations)

     nm = nm +1

#########################################################################
#                                                                      ##
# Determining & recording the optimal t-test, correlation & NSC value  ##
#                                                                      ##
#########################################################################

   ## t-test optimal threshold value ##
       ## KNN ##
        tot.error.cv.t = apply(knn.tcvA, 2, sum)
        optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
        tknn.outA[nm] = optimal.tthreshold

       tot.error.cv.t = apply(knn.tcvB, 2, sum)
       optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
       tknn.outB[nm] = optimal.tthreshold

        tot.error.cv.t = apply(knn.tcvC, 2, sum)
        optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
        tknn.outC[nm] = optimal.tthreshold

       ## SVM ##
        tot.error.cv.t = apply(error.tcvA, 2, sum)
        optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
        ttest.outA[nm] = optimal.tthreshold
```

```
         tot.error.cv.t = apply(error.tcvB, 2, sum)
         optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
         ttest.outB[nm] = optimal.tthreshold

         tot.error.cv.t = apply(error.tcvC, 2, sum)
         optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
         ttest.outC[nm] = optimal.tthreshold

     ## correlation optimal threshold value ##
        ## KNN ##
         tot.error.cv.cor = apply(knn.corcvA, 2, sum)
         optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
         corknn.outA[nm] = optimal.corthreshold

         tot.error.cv.cor = apply(knn.corcvB, 2, sum)
         optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
         corknn.outB[nm] = optimal.corthreshold

         tot.error.cv.cor = apply(knn.corcvC, 2, sum)
         optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
         corknn.outC[nm] = optimal.corthreshold
        ## SVM ##
         tot.error.cv.cor = apply(error.corcvA, 2, sum)
         optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
         cor.outA[nm] = optimal.corthreshold

         tot.error.cv.cor = apply(error.corcvB, 2, sum)
         optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
         cor.outB[nm] = optimal.corthreshold

         tot.error.cv.cor = apply(error.corcvC, 2, sum)
         optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
         cor.outC[nm] = optimal.corthreshold

    ## NSC optimal threshold ##
         tot.error.cv.nsc = apply(error.NSCcv, 2, sum)
         min.error=min(tot.error.cv.nsc)
         nsc.out[nm] = pamr.thr[max(which(tot.error.cv.nsc==min.error))]

    } #H
         print(nm)
  }  #A (end of nm CV)

##########################################################################
#                                                                       ##
# Returning the output of the thresholding values                       ##
#                                                                       ##
##########################################################################

    ## KNN frequency ##
      tA = table(tknn.outA)
      tB = table(tknn.outB)
      tC = table(tknn.outC)

      cA = table(corknn.outA)
      cB = table(corknn.outB)
      cC = table(corknn.outC)

    ## SVM frequency ##
      ttestA.freq = table(ttest.outA)
      ttestB.freq = table(ttest.outB)
      ttestC.freq = table(ttest.outC)

      corA.freq = table(cor.outA)
      corB.freq = table(cor.outB)
      corC.freq = table(cor.outC)

     ## NSC frequency ##
      NSC.freq = table(nsc.out)

   output = list("KNN ttestA freq"=tA, "KNN ttest B freq"=tB, "KNN ttestC
    freq"= tC,"KNN corA freq"=cA, "KNN corB freq"=cB, "KNN corC freq"=cC,
    "SVM ttestA freq"=ttestA.freq, "SVM ttest B freq"=ttestB.freq, "SVM
     ttestC freq"= ttestC.freq,"SVM corA freq"=corA.freq,"SVM corB freq"=
     corB.freq, "SVM corC freq"= corC.freq,"15 possible NSC
     thresholds"=pamr.thr,"NSC classifier freq" = NSC.freq)

   return(output)
 }


   ncol = ncol(data.xy)
   p = ncol-1
   N = nrow(data.xy)
   data.x = as.matrix(data.xy[,-ncol], nrow=N, ncol=p)
   yvec = data.xy[,ncol]
   data.y = yvec

  tknn.outA= tknn.outB = tknn.outC = matrix(rep(0),ncol=1,nrow=numsplits)
  corknn.outA=corknn.outB=corknn.outC=matrix(rep(0),ncol=1,nrow=numsplits)
  nscknn.outA=nscknn.outB=nscknn.outC=matrix(rep(0),ncol=1,nrow=numsplits)
  ttest.outA=ttest.outB=ttest.outC = matrix(rep(0),ncol=1,nrow=numsplits)
  cor.outA = cor.outB = cor.outC = matrix(rep(0),ncol=1,nrow=numsplits)
  nsc.outA = nsc.outB = nsc.outC = matrix(rep(0),ncol=1,nrow=numsplits)
  nsc.out = matrix(rep(0),ncol=1,nrow=numsplits)

#########################################################################
#                                                                      ##
# Obtaining the candidate corr. & NSC threshold values frm data.xy     ##
#                                                                      ##
#########################################################################
   ## Obtaining the 5 candidate correlation threshold values ##
    corvec = abs(cor(data.y,data.x))
    cor.thr  = seq(from=0.1, to=0.8*max(corvec), length=5)

   ## Obtain 15 candidate NSC threshold values from pamr package ##
     pam.x = t(data.x)
     pam.y = as.factor(data.y)
     pam.data = list(x=pam.x, y=pam.y)
     pamr.threshold = pamr.train(pam.data,n.threshold=30)$threshold
     pamr.thr = pamr.threshold[8:22]

#########################################################################
#                                                                      ##
# Performing LOOCV to find the optimal t-test,correlation & NSC values ##
#                                                                      ##
#########################################################################

## Splitting the data into training & test for LOOCV ##
   ind0 = which(yvec == 0)
   ind1 = which(yvec == 1)
   N0 = length(ind0)
   N1 = length(ind1)
   trainsize0 = floor(trfrac*N0)
   trainsize1 = floor(trfrac*N1)
   trainsize = trainsize0 + trainsize1
   testsize0 = N0 - trainsize0
   testsize1 = N1 - trainsize1
   testsize = testsize0 + testsize1

# while loop for repeatedly splitting data into training & test parts
   nm = 0

   while (nm<numsplits)  { #A
     ok = TRUE  # Intially the split is fine

     trainind0 = sample (ind0, trainsize0, replace=F)
     trainind1 = sample (ind1, trainsize1, replace=F)
     trainind = c(trainind0,trainind1)
     traindata.x = data.x[trainind,]
     traindata.y = data.y[trainind]
     traindata.xy = cbind(traindata.x, traindata.y)
     testdata.x = data.x[-trainind,]
     testdata.y = data.y[-trainind]

      # creating 3-dimensional arrays
      nscAr = array(0, c(nrow(traindata.x), p, length(pamr.thr)))
      ttestAr = array(0, c(nrow(traindata.x), p, length(t.testthr)))
      corAr = array(0, c(nrow(traindata.x), p, length(cor.thr)))

    # Note that we are currently inside the loop splitting the data into
training and test parts
       for (i in 1:nrow(traindata.x)) {           #B: LOOCV loop
          xtrain = traindata.x[-i,]
          ytrain = traindata.y[-i]
          xytrain = cbind(xtrain, ytrain)
          xtest = matrix(traindata.x[i,],nrow=1,ncol=p)
          ytest = traindata.y[i]


          ## Performing NSC thresholding ##
              if (ok == TRUE) {  #BB
               for (d in 1:length(pamr.thr)) {     #C
                  threshold = pamr.thr[d]
                  nsc = Pnsc(xytrain,threshold)
                  xindnsc = which(apply(nsc$nsc.omit,2,sum)>0)

                  if (length(xindnsc)==0) ok=FALSE
                  if (length(xindnsc)>0)   nscAr[i, xindnsc, d] =1
                 } #C
               } #BB


          ## Performing T-test thresholding ##
              if (ok == TRUE) {  #D
               for (j in 1:length(t.testthr)) { #E
                  threshold = t.testthr[j]
                  tout = Pttest(xytrain, threshold)
                  xindt = tout$Indices

                  if (length(xindt)==0) ok=FALSE
                  if (length(xindt)>0)   ttestAr[i, xindt, j] =1
                } #E
               } # D


          ## Performing Correlation thresholding ##
              if (ok == TRUE) {   #F
               for (j in 1:length(cor.thr)) { #G
                  threshold = cor.thr[j]
                  corout = Pcor(xytrain, threshold)
                  xindcor = corout$Indices

                  if (length(xindcor)==0) ok=FALSE
                  if (length(xindcor)>0)   corAr[i, xindcor, j] =1
                   }  #G
                } # F

         } #B (closing LOOCV)

        # Now LOOCV errors can be computed if ok still =TRUE
        if(ok==TRUE) { # H

        knn.tcvA= knn.tcvB= knn.tcvC= matrix(0, ncol = length(t.testthr),
nrow = nrow(traindata.x))
        knn.corcvA =knn.corcvB=knn.corcvC= matrix(0, ncol =
length(cor.thr), nrow = nrow(traindata.x))
        knn.nsccvA =knn.nsccvB=knn.nsccvC= matrix(0, ncol = 15, nrow =
nrow(traindata.x))

        error.tcvA = error.tcvB = error.tcvC = matrix(0, ncol =
length(t.testthr), nrow = nrow(traindata.x))
        error.corcvA = error.corcvB = error.corcvC = matrix(0, ncol =
length(cor.thr), nrow = nrow(traindata.x))
        error.nsccvA = error.nsccvB = error.nsccvC = matrix(0, ncol = 15,
nrow = nrow(traindata.x))

        error.NSCcv = matrix(0, ncol = 15, nrow = nrow(traindata.x))

        for (i in 1:nrow(traindata.x)) {           # I
           xtrain = traindata.x[-i,]
           ytrain = traindata.y[-i]
           xytrain = cbind(xtrain, ytrain)
           xtest = matrix(traindata.x[i,],nrow=1,ncol=p)
           ytest = traindata.y[i]


        ## Compute an error for every NSC threshold in pamr ##
            for (d in 1:length(pamr.thr)) {     # J
                threshold = pamr.thr[d]
                xindnsc = which(nscAr[i, ,d]==1)
                x.reducednsc =
matrix(xtrain[,xindnsc],nrow=nrow(xtrain),ncol=length(xindnsc))

             ## KNN ##
               out = knn(train = x.reducednsc, test =
traindata.x[i,xindnsc], cl = as.factor(ytrain), k=Kvec[1])
               knn.nsccvA[i, d] = (as.numeric(out)-1) != traindata.y[i]

               out = knn(train = x.reducednsc, test =
traindata.x[i,xindnsc], cl = as.factor(ytrain), k=Kvec[2])
               knn.nsccvB[i, d] = (as.numeric(out)-1) != traindata.y[i]
               out = knn(train = x.reducednsc, test =
traindata.x[i,xindnsc], cl = as.factor(ytrain), k=Kvec[3])
               knn.nsccvC[i, d] = (as.numeric(out)-1) != traindata.y[i]

             ## SVM ##
               out =
Psvm(x.reducednsc,ytrain,matrix(xtest[,xindnsc],nrow=1,ncol=length(xindns
c)),Cpar[1])
               error.nsccvA[i, d] = as.numeric((sign(out)+1)/2 != ytest)

               out = Psvm(x.reducednsc,ytrain,
matrix(xtest[,xindnsc],nrow=1,ncol=length(xindnsc)),Cpar[2])
               error.nsccvB[i, d] = as.numeric((sign(out)+1)/2 != ytest)

               out =
Psvm(x.reducednsc,ytrain,matrix(xtest[,xindnsc],nrow=1,ncol=length(xindns
c)),Cpar[3])
               error.nsccvC[i, d] = as.numeric((sign(out)+1)/2 != ytest)

             ## NSC ##
               out = Pnsc_class(xtrain, ytrain, xtest, delta=threshold)
               error.NSCcv[i, d] = sum(as.numeric(out!=  traindata.y[i]))

            } # J

        ## Compute an error for every threshold in t-test ##
            for (j in 1:length(t.testthr)) {     # K
                threshold = t.testthr[j]
                xindt = which(ttestAr[i, ,j]==1)
                x.reducedt =
matrix(xtrain[,xindt],nrow=nrow(xtrain),ncol=length(xindt))

             ## KNN ##
               out = knn(train = x.reducedt, test = traindata.x[i,xindt],
cl = as.factor(ytrain), k=Kvec[1])
               knn.tcvA[i, j] = (as.numeric(out)-1) != traindata.y[i]

               out = knn(train = x.reducedt, test = traindata.x[i,xindt],
cl = as.factor(ytrain), k=Kvec[2])
               knn.tcvB[i, j] = (as.numeric(out)-1) != traindata.y[i]

               out = knn(train = x.reducedt, test = traindata.x[i,xindt],
cl = as.factor(ytrain), k=Kvec[3])
               knn.tcvC[i, j] = (as.numeric(out)-1) != traindata.y[i]


             ## SVM ##
               out =
Psvm(x.reducedt,ytrain,matrix(xtest[,xindt],nrow=1,ncol=length(xindt)),Cp
ar[1])
               error.tcvA[i, j] = as.numeric((sign(out)+1)/2 != ytest)
#sign(fvalues) = -1 or 1

               out =
Psvm(x.reducedt,ytrain,matrix(xtest[,xindt],nrow=1,ncol=length(xindt)),Cp
ar[2])
               error.tcvB[i, j] = as.numeric((sign(out)+1)/2 != ytest)

               out =
Psvm(x.reducedt,ytrain,matrix(xtest[,xindt],nrow=1,ncol=length(xindt)),Cp
ar[3])
```

```
               error.tcvC[i, j] = as.numeric((sign(out)+1)/2 != ytest)
             } # K

        ## Compute an error for every threshold in corr ##
            for (j in 1:length(cor.thr)) {     # L
                threshold = cor.thr[j]
                xindcor = which(corAr[i, ,j]==1)
                x.reducedcor =
matrix(xtrain[,xindcor],nrow=nrow(xtrain),ncol=length(xindcor))

             ## KNN ##
               out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[1])
               knn.corcvA[i, j] = (as.numeric(out)-1) != traindata.y[i]

               out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[2])
               knn.corcvB[i, j] = (as.numeric(out)-1) != traindata.y[i]

               out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[3])
               knn.corcvC[i, j] = (as.numeric(out)-1) != traindata.y[i]


             ## SVM ##
               out =
Psvm(x.reducedcor,ytrain,matrix(xtest[,xindcor],nrow=1,ncol=length(xindco
r)),Cpar[1])
               error.corcvA[i, j] = as.numeric((sign(out)+1)/2 != ytest)

               out = Psvm(x.reducedcor,ytrain,
matrix(xtest[,xindcor],nrow=1,ncol=length(xindcor)),Cpar[2])
               error.corcvB[i, j] = as.numeric((sign(out)+1)/2 != ytest)

               out = Psvm(x.reducedcor,ytrain,
matrix(xtest[,xindcor],nrow=1,ncol=length(xindcor)),Cpar[3])
               error.corcvC[i, j] =as.numeric((sign(out)+1)/2 != ytest)
            } # L


       } #I (end of error computations)


          nm = nm +1

       ## t-test optimal ##

             ## KNN ##
              tot.error.cv.t = apply(knn.tcvA, 2, sum)
              optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
              tknn.outA[nm] = optimal.tthreshold

             tot.error.cv.t = apply(knn.tcvB, 2, sum)
             optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
             tknn.outB[nm] = optimal.tthreshold

              tot.error.cv.t = apply(knn.tcvC, 2, sum)
              optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
              tknn.outC[nm] = optimal.tthreshold

             ## SVM ##
```

```
              tot.error.cv.t = apply(error.tcvA, 2, sum)
              optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
              ttest.outA[nm] = optimal.tthreshold

              tot.error.cv.t = apply(error.tcvB, 2, sum)
              optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
              ttest.outB[nm] = optimal.tthreshold

              tot.error.cv.t = apply(error.tcvC, 2, sum)
              optimal.tthreshold = t.testthr[which.min(tot.error.cv.t)]
              ttest.outC[nm] = optimal.tthreshold

       ## correlation optimal threshold ##
             ## KNN ##
              tot.error.cv.cor = apply(knn.corcvA, 2, sum)
              optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
              corknn.outA[nm] = optimal.corthreshold

              tot.error.cv.cor = apply(knn.corcvB, 2, sum)
              optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
              corknn.outB[nm] = optimal.corthreshold

              tot.error.cv.cor = apply(knn.corcvC, 2, sum)
              optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
              corknn.outC[nm] = optimal.corthreshold


             ## SVM ##
              tot.error.cv.cor = apply(error.corcvA, 2, sum)
              optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
              cor.outA[nm] = optimal.corthreshold

              tot.error.cv.cor = apply(error.corcvB, 2, sum)
              optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
              cor.outB[nm] = optimal.corthreshold

              tot.error.cv.cor = apply(error.corcvC, 2, sum)
              optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
              cor.outC[nm] = optimal.corthreshold

       ## NSC optimal threshold ##

             ## KNN ##
              tot.error.cv.nsc = apply(knn.nsccvA, 2, sum)
                 min.error=min(tot.error.cv.nsc)
              nscknn.outA[nm] =
pamr.thr[max(which(tot.error.cv.nsc==min.error))]

              tot.error.cv.nsc = apply(knn.nsccvB, 2, sum)
                 min.error=min(tot.error.cv.nsc)
              nscknn.outB[nm] =
pamr.thr[max(which(tot.error.cv.nsc==min.error))]

              tot.error.cv.nsc = apply(knn.nsccvC, 2, sum)
                  min.error=min(tot.error.cv.nsc)
              nscknn.outC[nm] =
pamr.thr[max(which(tot.error.cv.nsc==min.error))]

             ## SVM ##
              tot.error.cv.nsc = apply(error.nsccvA, 2, sum)   #cv for
single split
```

```
              min.error=min(tot.error.cv.nsc)

nsc.outA[nm]=pamr.thr[max(which(tot.error.cv.nsc==min.error))]

              tot.error.cv.nsc = apply(error.nsccvB, 2, sum)
              min.error=min(tot.error.cv.nsc)

nsc.outB[nm]=pamr.thr[max(which(tot.error.cv.nsc==min.error))]

              tot.error.cv.nsc = apply(error.nsccvC, 2, sum)
              min.error=min(tot.error.cv.nsc)

nsc.outC[nm]=pamr.thr[max(which(tot.error.cv.nsc==min.error))]

             ## NSC ##
              tot.error.cv.nsc = apply(error.NSCcv, 2, sum)
              min.error=min(tot.error.cv.nsc)
              nsc.out[nm] =
pamr.thr[max(which(tot.error.cv.nsc==min.error))]

         } #H
        print(nm)
     }  #A (end of nm CV)

##########################################################################
#                                                                       ##
# Returning the selection frequencies of the candidate thresh. values  ##
#                                                                       ##
##########################################################################
 ## KNN classifier's Selection Frequencies ##
    tA = table(tknn.outA)
    tB = table(tknn.outB)
    tC = table(tknn.outC)

    cA = table(corknn.outA)
    cB = table(corknn.outB)
    cC = table(corknn.outC)

    nA = table(nscknn.outA)
    nB = table(nscknn.outB)
    nC = table(nscknn.outC)

 ## SVM classifier's selection Frequencies ##
     ttestA.freq = table(ttest.outA)
     ttestB.freq = table(ttest.outB)
     ttestC.freq = table(ttest.outC)

     corA.freq = table(cor.outA)
     corB.freq = table(cor.outB)
     corC.freq = table(cor.outC)

     nscA.freq = table(nsc.outA)
     nscB.freq = table(nsc.outB)
     nscC.freq = table(nsc.outC)

  ## NSC classifier's selection frequencies ##
     NSC.freq = table(nsc.out)

## Outputting the results ##
 output= list("KNN ttestA freq"=tA, "KNN ttest B freq"=tB,
         "KNN ttestC freq"= tC, "KNN corA freq"=cA,"KNN corB freq"=cB,
         "KNN corC freq"=cC, "15 possible NSC thresholds"=pamr.thr,
         "KNN nscA freq"=nA,"KNN nscB freq"=nB,"KNN nscC freq"=nC,

         "SVM ttestA freq"=ttestA.freq, "SVM ttest B freq"=ttestB.freq,
         "SVM ttestC freq"= ttestC.freq,"SVM corA freq"=corA.freq,
         "SVM corB freq"=corB.freq, "SVM corC freq"= corC.freq,
         "15 possible NSC thresholds"=pamr.thr,
         "SVM nscA freq" = nscA.freq, "SVM nscB freq" = nscB.freq,
         "SVM nscC freq" = nscC.freq,"NSC classifier freq" = NSC.freq)

  return(output)
}
```

B.23 Using LOOCV to determine optimal threshold values for multi-class data reported in Section 5.6

```
> fix(Pthreshold_mult)
```

```
function(data.xy, Kvec, numsplits, trfrac) {
    # Kvec= vector containing 3 possible values for the K in KNN

##########################################################################
#                                                                       ##
# This program determines the threshold values to be used in the:      ##
#    a) Correlation thresholding  &                                     ##
#    b) NSC VS procedure                                                ##
#                                                                       ##
##########################################################################

## Obtain matrix of input values & binary response vector from data.xy ##
   ncol = ncol(data.xy)
   p = ncol-1
   N = nrow(data.xy)
   data.x = as.matrix(data.xy[,-ncol], nrow=N, ncol=p)
   yvec = data.xy[,ncol]
   data.y = yvec

##########################################################################
#                                                                       ##
# Obtaining the candidate corr. & NSC threshold values frm data.xy     ##
#                                                                       ##
##########################################################################

  # Obtaining the 5 candidate correlation threshold values
    corvec = abs(cor(data.y,data.x))
    cor.thr  = seq(from=0.1, to=0.9*max(corvec), length=5)

  # Obtaining 15 candidate NSC threshold values from pamr package
    pam.x = t(data.x)
    pam.y = as.factor(data.y)
    pam.data = list(x=pam.x, y=pam.y)
    pamr.threshold = pamr.train(pam.data,n.threshold=30)$threshold      #
    pamr.thr = pamr.threshold[8:22]

   corknn.outA=corknn.outB=corknn.outC=matrix(rep(0),ncol=1,nrow=numsplits)
   nscknn.outA=nscknn.outB=nscknn.outC=matrix(rep(0),ncol=1,nrow=numsplits)
   nsc.out = matrix(rep(0),ncol=1,nrow=numsplits)
```

```
##########################################################################
#                                                                       ##
# Performing LOOCV to find the optimal correlation & NSC values         ##
#                                                                       ##
##########################################################################

## Splitting the data into training & test for LOOCV ##

   ind1 = which(yvec == 1)
   ind2 = which(yvec == 2)
   ind3 = which(yvec == 3)
   ind4 = which(yvec == 4)
   N1= length(ind1); N2= length(ind2); N3= length(ind3); N4= length(ind4)

   trainsize1 = floor(trfrac*N1)
   trainsize2 = floor(trfrac*N2)
   trainsize3 = floor(trfrac*N3)
   trainsize4 = floor(trfrac*N4)
   trainsize = trainsize1 + trainsize2 + trainsize3 + trainsize4

   testsize1 = N1 - trainsize1
   testsize2 = N2 - trainsize2
   testsize3 = N3 - trainsize3
   testsize4 = N4 - trainsize4
   testsize = testsize1 + testsize2+ testsize3+ testsize4

   # start of while loop for repeatedly splitting data into train& test
   nm = 0
    while (nm<numsplits)  { #A
    ok = TRUE  # Intially the split is fine

     print(nm)
     trainind1 = sample (ind1, trainsize1, replace=F)
       trainind2 = sample (ind2, trainsize2, replace=F)
       trainind3 = sample (ind3, trainsize3, replace=F)
       trainind4 = sample (ind4, trainsize4, replace=F)
       trainind = c(trainind1,trainind2,trainind3,trainind4)
       traindata.x = data.x[trainind,]
       traindata.y = data.y[trainind]
       traindata.xy = cbind(traindata.x, traindata.y)
       testdata.x = data.x[-trainind,]
       testdata.y = data.y[-trainind]

    # creating 3-dimensional arrays
     nscAr = array(0, c(nrow(traindata.x), p, length(pamr.thr)))
     corAr = array(0, c(nrow(traindata.x), p, length(cor.thr)))


 # Note that we are currently inside the loop splitting the data into
training and test parts
       for (i in 1:nrow(traindata.x)) {           #B: LOOCV loop
            xtrain = traindata.x[-i,]
            ytrain = traindata.y[-i]
            xytrain = cbind(xtrain, ytrain)
            xtest = matrix(traindata.x[i,],nrow=1,ncol=p)
            ytest = traindata.y[i]


              ## Performing NSC thresholding
                  if (ok == TRUE) {  #C
                   for (d in 1:length(pamr.thr)) {     #D
```

```
                       threshold = pamr.thr[d]
                       nsc = Pnsc_mult(xytrain,threshold)
                       xindnsc = which(apply(nsc$nsc.omit,2,sum)>0)

                       if (length(xindnsc)==0) ok=FALSE
                       if (length(xindnsc)>0)   nscAr[i, xindnsc, d] =1
                     } #D
                   } #C



               ## Performing Correlation thresholding ##
                   if (ok == TRUE) {  #E
                    for (j in 1:length(cor.thr)) { #F
                       threshold = cor.thr[j]
                       corout = Pcor(xytrain, threshold)
                       xindcor = corout$Indices

                      if (length(xindcor)==0) ok=FALSE
                      if (length(xindcor)>0)   corAr[i, xindcor, j] =1
                      }  #F
                    } # E

           } #B (closing LOOCV)

         # Now LOOCV errors can be computed if ok still =TRUE
         if(ok==TRUE) { # G

      knn.corcvA =knn.corcvB=knn.corcvC=
          matrix(0, ncol = length(cor.thr), nrow = nrow(traindata.x))
      knn.nsccvA =knn.nsccvB=knn.nsccvC=
          matrix(0, ncol = 15, nrow = nrow(traindata.x))
      error.NSCcv = matrix(0, ncol = 15, nrow = nrow(traindata.x))

       for (i in 1:nrow(traindata.x)) {           # H
           xtrain = traindata.x[-i,]
           ytrain = traindata.y[-i]
           xytrain = cbind(xtrain, ytrain)
           xtest = matrix(traindata.x[i,],nrow=1,ncol=p)
           ytest = traindata.y[i]


        ## Compute an error for every threshold in pamr
           for (d in 1:length(pamr.thr)) {     # J
               threshold = pamr.thr[d]
               xindnsc = which(nscAr[i, ,d]==1)
               x.reducednsc=
matrix(xtrain[,xindnsc],nrow=nrow(xtrain),ncol=length(xindnsc))

             ## KNN ##
               out = knn(train = x.reducednsc, test =
traindata.x[i,xindnsc], cl = as.factor(ytrain), k=Kvec[1])
               knn.nsccvA[i, d] = (as.numeric(out)-1) != traindata.y[i]

               out = knn(train = x.reducednsc, test =
traindata.x[i,xindnsc], cl = as.factor(ytrain), k=Kvec[2])
               knn.nsccvB[i, d] = (as.numeric(out)-1) != traindata.y[i]

               out = knn(train = x.reducednsc, test =
traindata.x[i,xindnsc], cl = as.factor(ytrain), k=Kvec[3])
               knn.nsccvC[i, d] = (as.numeric(out)-1) != traindata.y[i]
          ## NSC ##
           out = Pnsc_class_mult(xtrain, ytrain, xtest, delta=threshold)
             error.NSCcv[i, d] = sum(as.numeric(out!=  traindata.y[i]))

           } # J

       ## Compute an error for every threshold in corr
          for (j in 1:length(cor.thr)) {      # L
              threshold = cor.thr[j]
              xindcor = which(corAr[i, ,j]==1)
              x.reducedcor =
matrix(xtrain[,xindcor],nrow=nrow(xtrain),ncol=length(xindcor))

             ## KNN ##
             out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[1])
             knn.corcvA[i, j] = (as.numeric(out)-1) != traindata.y[i]

             out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[2])
             knn.corcvB[i, j] = (as.numeric(out)-1) != traindata.y[i]

             out = knn(train = x.reducedcor, test =
traindata.x[i,xindcor], cl = as.factor(ytrain), k=Kvec[3])
             knn.corcvC[i, j] = (as.numeric(out)-1) != traindata.y[i]
           } # L


      } #H (end of error computations)


         nm = nm +1

      ## Correlation optimal threshold ##
         ## KNN ##
            tot.error.cv.cor = apply(knn.corcvA, 2, sum)
            optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
            corknn.outA[nm] = optimal.corthreshold

            tot.error.cv.cor = apply(knn.corcvB, 2, sum)
            optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
            corknn.outB[nm] = optimal.corthreshold

            tot.error.cv.cor = apply(knn.corcvC, 2, sum)
            optimal.corthreshold = cor.thr[which.min(tot.error.cv.cor)]
            corknn.outC[nm] = optimal.corthreshold

     ## NSC optimal threshold ##

         ## KNN ##
            tot.error.cv.nsc = apply(knn.nsccvA, 2, sum)
            min.error=min(tot.error.cv.nsc)
            nscknn.outA[nm] =
                pamr.thr[max(which(tot.error.cv.nsc==min.error))]

            tot.error.cv.nsc = apply(knn.nsccvB, 2, sum)
            min.error=min(tot.error.cv.nsc)
            nscknn.outB[nm] =
                 pamr.thr[max(which(tot.error.cv.nsc==min.error))]
              tot.error.cv.nsc = apply(knn.nsccvC, 2, sum)
              min.error=min(tot.error.cv.nsc)
              nscknn.outC[nm] =
                   pamr.thr[max(which(tot.error.cv.nsc==min.error))]

           ## NSC ##
              tot.error.cv.nsc = apply(error.NSCcv, 2, sum)
              min.error=min(tot.error.cv.nsc)
              nsc.out[nm] =
                  pamr.thr[max(which(tot.error.cv.nsc==min.error))]

          } #G
      }  #A (end of nm CV)

##########################################################################
#                                                                       ##
# Returning the output of the thresholding values                       ##
#                                                                       ##
##########################################################################
   ## KNN frequency ##
      cA = table(corknn.outA)
      cB = table(corknn.outB)
      cC = table(corknn.outC)
      nA = table(nscknn.outA)
      nB = table(nscknn.outB)
      nC = table(nscknn.outC)

   ## NSC ##
       NSC.freq = table(nsc.out)

    output = list("5 possible correlation values"=cor.thr,
              "KNN corA freq"=cA, "KNN corB freq"=cB, "KNN corC freq"=cC,
              "15 possible NSC thresholds"=pamr.thr,
              "KNN nscA freq"=nA,"KNN nscB freq"=nB, "KNN nscC freq"=nC,
              "NSC classifier freq" = NSC.freq)

  return(output)
}
```

B.24 Applying the classification procedures to the binary data sets

```
>  fix(PMain)
```

```
function(data.xy, numsplits, trfrac, Kvec, Cpar, pcathresh, t.testthr,
corthr, nsc.thr) {
  #Kvec is a vector of 3 values for K in KNN
  #Cpar is a vector of 3 values for the C parameter in RBF SVM
  #t.testthr is a vector of 6 values
  # corthr is a vector of 6 values
  # nsc.thr is a vector of 1 values (selected from NSC)

##########################################################################
#                                                                       ##
# This program analyses a given binary data set using different         ##
# versions of the KNN,fastKNN, SVM, LDA, DLDA & NSC classifier          ##
#                                                                       ##
# The following procedures are applied:                                 ##
# 1. KNN/ SVM using all the input variables                             ##
# 2. fastkNN/ SVM using features extracted by means of fastKNN          ##
# 3. KNN/ SVM using only variables selected by t-test thresholding      ##
# 4. KNN/ SVM using only variables selected by correlation thresholding##
# 5. fastkNN using features extracted based only on correlation thresh.##
# 6. KNN/ SVM using ordinary PCs extracted from the full data set      ##
# 7. KNN/ SVM using supervised PCs based on t-test thresholding        ##
# 8. KNN/ SVM  using supervised PCs based on correlation thresholding  ##
# 9. kNN/ SVM using NSC selected features from opt. delta threshlold   ##
#                                                                      ##
# 10. LDA using NSC selected features from optimal delta threshold     ##
# 11. NSC using NSC select features from opt. delta threshold          ##
# 12. DLDA using all the input variables                               ##
# 13. DLDA using NSC selected features from optimal delta threshold    ##
#                                                                      ##
########################################################################

########################################################################
#                                                                      ##
# Function to obtain the mode                                          ##
#                                                                      ##
########################################################################
 PMode = function(x) {
    ux = unique(x)
    ux[which.max(tabulate(match(x, ux)))]
    }

########################################################################
#                                                                      ##
# Data properties, Pre-processing the data &                           ##
# Obtain the row indices identifying the Y = 0 & Y = 1 data cases      ##
#                                                                      ##
########################################################################

 ncol = ncol(data.xy)
 p = ncol-1
 N = nrow(data.xy)
 data.x = as.matrix(data.xy[,-ncol])
 yvec = data.xy[,ncol]
 data.y = yvec
 ind0 = which(yvec == 0)
 ind1 = which(yvec == 1)
 N0 = length(ind0)
 N1 = length(ind1)

 knnerrormatA= knnerrormatB=knnerrormatC=matrix(0,nrow=numsplits,ncol=9)
 knnFNRmatA= knnFNRmatB=knnFNRmatC =matrix(0, nrow =numsplits, ncol =9)

 svmerrormatA =svmerrormatB=svmerrormatC=matrix(0,nrow=numsplits,ncol= 9)
 svmFNRmatA = svmFNRmatB = svmFNRmatC =matrix(0, nrow=numsplits, ncol= 9)

 LDAerrormat = matrix(0, nrow = numsplits, ncol = 4)
 LDAFNRmat = matrix(0, nrow = numsplits, ncol = 4)

 no.featA =matrix(0, nrow=numsplits, ncol=9)
 no.featB = matrix(0, nrow=numsplits, ncol=9)
 no.featC = matrix(0, nrow=numsplits, ncol=9)
 no.featsvmA= no.featsvmB= no.featsvmC= matrix(0, nrow=numsplits, ncol=9)
 no.featLDA = matrix(0, nrow=numsplits, ncol=4)

 # Variables indices selected by ##
 rvsmatA = rvsmatB = rvsmatC = matrix(0, nrow=numsplits, ncol=p)  #corr
 tvsmatA =  tvsmatB = tvsmatC = matrix(0, nrow=numsplits, ncol=p) #t-test
 SVMrvsmatA = SVMrvsmatB = SVMrvsmatC =matrix(0, nrow=numsplits, ncol=p)
    #genes index selected in corr
 SVMtvsmatA =  SVMtvsmatB = SVMtvsmatC=matrix(0, nrow=numsplits, ncol=p)
    #genes index selected in t-test

 nscvsmat = matrix(0, nrow=numsplits, ncol=p) #nsc selected genes index

 trainsize0 = floor(trfrac*N0)
 trainsize1 = floor(trfrac*N1)
 trainsize = trainsize0 + trainsize1
 testsize0 = N0 - trainsize0
 testsize1 = N1 - trainsize1
 testsize = testsize0 + testsize1

 for (nm in 1:numsplits)  {
    print(nm)
    trainind0 = sample (ind0, trainsize0, replace=F)
    trainind1 = sample (ind1, trainsize1, replace=F)
    trainind = c(trainind0,trainind1)
    traindata.x = data.x[trainind,]
    traindata.y = data.y[trainind]
    traindata.xy = cbind(traindata.x, traindata.y)
    testdata.x = data.x[-trainind,]
    testdata.y = data.y[-trainind]

#########################################################################
#                                                                      ##
# METHOD 1: Applying ordinary kNN/ SVM using all the input variables   ##
#                                                                      ##
#########################################################################


  no.featA[nm,1] =no.featB[nm,1]=no.featC[nm,1]= ncol(traindata.x)

    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatA[nm, 1] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,1] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[2], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatB[nm, 1] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,1] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[3], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatC[nm, 1] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,1] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

##Note that in SVM the sign(fvalues) = -1 or 1 & need to be transformed##

no.featsvmA[nm,1]=no.featsvmB[nm,1]=no.featsvmC[nm,1]=ncol(traindata.x)

    out = Psvm(traindata.x, traindata.y, testdata.x,Cpar[1])
    svmerrormatA[nm, 1] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 1] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(traindata.x,traindata.y,testdata.x,Cpar[2])
    svmerrormatB[nm,1] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 1] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(traindata.x,traindata.y,testdata.x,Cpar[3])
    svmerrormatC[nm,1] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 1] = length(which((testdata.y-(sign(out)+1)/2)==1))

#########################################################################
#                                                                      ##
# METHOD 2: Applying the fastKNN classifier to the fastKNN extracted   ##
#           Applying SVM classifier to fastKNN features using K=Kvec[3]##
#                                                                      ##
#########################################################################

    extracted.features =PfastKNNset (traindata.x,traindata.y,Kvec[1],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featA[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[1])$class
    knnerrormatA[nm, 2] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,2] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    extracted.features =PfastKNNset (traindata.x,traindata.y,Kvec[2],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featB[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[2])$class
    knnerrormatB[nm, 2] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,2] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    extracted.features =PfastKNNset (traindata.x,traindata.y,Kvec[3],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featC[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[3])$class
    knnerrormatC[nm, 2] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,2] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

 no.featsvmA[nm,2]=no.featsvmB[nm,2]=no.featsvmC[nm,2] = ncol(extr.train)
    out = Psvm(extr.train,traindata.y, extr.test,Cpar[1])
    svmerrormatA[nm,2] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 2] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(extr.train,traindata.y, extr.test,Cpar[2])
    svmerrormatB[nm,2] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 2] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(extr.train,traindata.y, extr.test,Cpar[3])
    svmerrormatC[nm,2] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 2] = length(which((testdata.y-(sign(out)+1)/2)==1))
```

```
##########################################################################
#                                                                       ##
# METHOD 3: Applying KNN & SVM to the t-test selected variables         ##
#                                                                       ##
##########################################################################

    tout = Pttest(traindata.xy, t.testthr[1])
    xindtknnA = tout$Indices
    x.reducedtknnA = traindata.x[, xindtknnA]
    tvsmatA[nm,xindtknnA] = 1
    no.featA[nm,3] =length(xindtknnA)
    out = knn(traindata.x[,xindtknnA], testdata.x[,xindtknnA],
as.factor(traindata.y), k = Kvec[1], l = 0, prob = FALSE, use.all =
FALSE)
    knnerrormatA[nm, 3] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,3] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    tout = Pttest(traindata.xy, t.testthr[2])
    xindtknnB= tout$Indices
    x.reducedtknnB = traindata.x[, xindtknnB]
    tvsmatB[nm,xindtknnB] = 1
    no.featB[nm,3] =length(xindtknnB)
    out = knn(traindata.x[,xindtknnB], testdata.x[,xindtknnB],
as.factor(traindata.y), k = Kvec[2], l = 0, prob = FALSE, use.all =
FALSE)
    knnerrormatB[nm, 3] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,3] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    tout = Pttest(traindata.xy, t.testthr[3])
    xindtknnC= tout$Indices
    x.reducedtknnC = traindata.x[, xindtknnC]
    tvsmatC[nm,xindtknnC] = 1
    no.featC[nm,3] = length(xindtknnC)
    out = knn(traindata.x[,xindtknnC], testdata.x[,xindtknnC],
as.factor(traindata.y), k = Kvec[3], l = 0, prob = FALSE, use.all =
FALSE)
    knnerrormatC[nm, 3] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,3] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    tout = Pttest(traindata.xy, t.testthr[4])
    xindtsvmA = tout$Indices
    x.reducedtsvmA = traindata.x[, xindtsvmA]
    SVMtvsmatA[nm,xindtsvmA] = 1
    no.featsvmA[nm,3]= length(xindtsvmA)
    out = Psvm(x.reducedtsvmA,traindata.y,testdata.x[,xindtsvmA],Cpar[1])
    svmerrormatA[nm,3] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 3] = length(which((testdata.y-(sign(out)+1)/2)==1))

    tout = Pttest(traindata.xy, t.testthr[5])
    xindtsvmB = tout$Indices
    x.reducedtsvmB = traindata.x[, xindtsvmB]
    SVMtvsmatB[nm,xindtsvmB] = 1
    no.featsvmB[nm,3]= length(xindtsvmB)
    out = Psvm(x.reducedtsvmB,traindata.y,testdata.x[,xindtsvmB],Cpar[2])
    svmerrormatB[nm,3] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 3] = length(which((testdata.y-(sign(out)+1)/2)==1))

    tout = Pttest(traindata.xy, t.testthr[6])
    xindtsvmC = tout$Indices
    x.reducedtsvmC = traindata.x[, xindtsvmC]
    SVMtvsmatC[nm,xindtsvmC] = 1
    no.featsvmC[nm,3]= length(xindtsvmC)
    out = Psvm(x.reducedtsvmC,traindata.y,testdata.x[,xindtsvmC],Cpar[3])
    svmerrormatC[nm,3] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 3] = length(which((testdata.y-(sign(out)+1)/2)==1))

#########################################################################
#                                                                      ##
# METHOD 4: Applying KNN & SVM to the corr. thresholding selected vars ##
#                                                                      ##
#########################################################################
    corout = Pcor(traindata.xy, corthr[1])
    xindcorknnA = corout$Indices
    x.reducedcorknnA = traindata.x[, xindcorknnA]
    rvsmatA[nm,xindcorknnA] = 1
    no.featA[nm,4] = length(xindcorknnA)
    out = knn(traindata.x[,xindcorknnA], testdata.x[,xindcorknnA],
as.factor(traindata.y), k =Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatA[nm, 4] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,4] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    corout = Pcor(traindata.xy, corthr[2])
    xindcorknnB = corout$Indices
    x.reducedcorknnB = traindata.x[, xindcorknnB]
    rvsmatB[nm,xindcorknnB] = 1
    no.featB[nm,4] = length(xindcorknnB)
    out = knn(traindata.x[,xindcorknnB], testdata.x[,xindcorknnB],
as.factor(traindata.y), k = Kvec[2], l =0, prob = FALSE, use.all = FALSE)
    knnerrormatB[nm, 4] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,4] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    corout = Pcor(traindata.xy, corthr[3])
    xindcorknnC = corout$Indices
    x.reducedcorknnC = traindata.x[, xindcorknnC]
    rvsmatC[nm,xindcorknnC] = 1
    no.featC[nm,4] = length(xindcorknnC)
    out = knn(traindata.x[,xindcorknnC], testdata.x[,xindcorknnC],
as.factor(traindata.y), k = Kvec[3], l =0, prob = FALSE, use.all = FALSE)
    knnerrormatC[nm, 4] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,4] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

   corout = Pcor(traindata.xy, corthr[4])
   xindcorsvmA = corout$Indices
   x.reducedcorsvmA= traindata.x[, xindcorsvmA]
   SVMrvsmatA[nm,xindcorsvmA] = 1
   no.featsvmA[nm,4] = length(xindcorsvmA)
   out =Psvm(x.reducedcorsvmA,traindata.y,testdata.x[,xindcorsvmA],
        Cpar[1])
    svmerrormatA[nm,4] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 4] = length(which((testdata.y-(sign(out)+1)/2)==1))

    corout = Pcor(traindata.xy, corthr[5])
    xindcorsvmB = corout$Indices
    x.reducedcorsvmB= traindata.x[, xindcorsvmB]
    SVMrvsmatB[nm,xindcorsvmB] = 1
    no.featsvmB[nm,4] = length(xindcorsvmB)
    out = Psvm(x.reducedcorsvmB, traindata.y, testdata.x[,xindcorsvmB],
Cpar[2])
    svmerrormatB[nm,4] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 4] = length(which((testdata.y-(sign(out)+1)/2)==1))

    corout = Pcor(traindata.xy, corthr[6])
    xindcorsvmC = corout$Indices
    x.reducedcorsvmC= traindata.x[, xindcorsvmC]
    SVMrvsmatC[nm,xindcorsvmC] = 1
    no.featsvmC[nm,4] = length(xindcorsvmC)
    out = Psvm(x.reducedcorsvmC, traindata.y, testdata.x[,xindcorsvmC],
Cpar[3])
    svmerrormatC[nm,4] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 4] = length(which((testdata.y-(sign(out)+1)/2)==1))

########################################################################
#                                                                    ##
# METHOD 5: Applying the fastKNN classifier using the fastKNN features ##
#           extracted from the correlation thresholded data set        ##
#                                                                    ##
########################################################################
    extracted.features =PfastKNNset
(traindata.x[,xindcorknnA],traindata.y,Kvec[1], testdata.x[,xindcorknnA])
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featA[nm,5] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[1])$class
    knnerrormatA[nm, 5] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,5] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    extracted.features =PfastKNNset
(traindata.x[,xindcorknnB],traindata.y,Kvec[2], testdata.x[,xindcorknnB])
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featB[nm,5] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[2])$class
    knnerrormatB[nm, 5] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,5] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    extracted.features =PfastKNNset
(traindata.x[,xindcorknnC],traindata.y,Kvec[3], testdata.x[,xindcorknnC])
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featC[nm,5] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[3])$class
    knnerrormatC[nm, 5] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,5] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

########################################################################
#                                                                    ##
# METHOD 6: Applying KNN & SVM to the PC extracted from the full data  ##
#                                                                    ##
########################################################################
    pcaout = preProcess(data.frame(traindata.x), method=c("pca"), thresh
= pcathresh)
```

```
    number.of.comp = pcaout$numComp
    pcamat.train = traindata.x%*%pcaout$rotation
    pcamat.test = testdata.x%*%pcaout$rotation

    no.featA[nm,6]=no.featB[nm,6]=no.featC[nm,6] = number.of.comp

    out = knn(pcamat.train, pcamat.test, as.factor(traindata.y), k =
Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatA[nm, 6] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,6]= length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(pcamat.train, pcamat.test, as.factor(traindata.y), k =
Kvec[2], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatB[nm, 6] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,6] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(pcamat.train, pcamat.test, as.factor(traindata.y), k =
Kvec[3], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatC[nm, 6] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,6] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    no.featsvmA[nm,6] =no.featsvmB[nm,6]=no.featsvmC[nm,6]=number.of.comp

    out = Psvm(pcamat.train, traindata.y, pcamat.test, Cpar[1])
    svmerrormatA[nm,6] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 6] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(pcamat.train, traindata.y, pcamat.test, Cpar[2])
    svmerrormatB[nm,6] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 6] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(pcamat.train, traindata.y, pcamat.test, Cpar[3])
    svmerrormatC[nm,6] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 6] = length(which((testdata.y-(sign(out)+1)/2)==1))

#########################################################################
#                                                                     ##
# METHOD 7: Applying KNN &SVM to the SPCs based on t-test thresholding ##
#                                                                     ##
#########################################################################
    spcaout = preProcess(data.frame(x.reducedtknnA), method = c("pca"),
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindtknnA] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindtknnA] %*% spcaout$rotation
    no.featA[nm,7] = snumber.of.comp
    out = knn(spcamat.train, spcamat.test, as.factor(traindata.y), k =
Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatA[nm, 7] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm, 7] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    spcaout = preProcess(data.frame(x.reducedtknnB), method = c("pca"),
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindtknnB] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindtknnB] %*% spcaout$rotation
    no.featB[nm,7]= snumber.of.comp
    out = knn(spcamat.train, spcamat.test, as.factor(traindata.y), k =
Kvec[2], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatB[nm, 7] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm, 7] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    spcaout = preProcess(data.frame(x.reducedtknnC), method = c("pca"),
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindtknnC] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindtknnC] %*% spcaout$rotation
    no.featC[nm,7]= snumber.of.comp
    out = knn(spcamat.train, spcamat.test, as.factor(traindata.y), k =
Kvec[3], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatC[nm, 7] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm, 7] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    spcaout = preProcess(data.frame(x.reducedtsvmA), method = c("pca"),
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindtsvmA] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindtsvmA] %*% spcaout$rotation
    no.featsvmA[nm,7] = snumber.of.comp
    out = Psvm(spcamat.train, traindata.y, spcamat.test, Cpar[1])
    svmerrormatA[nm,7] = length(which((sign(out)+1)/2 != testdata.y))
#sign(fvalues) = -1 or 1
    svmFNRmatA[nm, 7] = length(which((testdata.y-(sign(out)+1)/2)==1))

    spcaout = preProcess(data.frame(x.reducedtsvmB), method = c("pca"),
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindtsvmB] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindtsvmB] %*% spcaout$rotation
    no.featsvmB[nm,7] = snumber.of.comp
    out = Psvm(spcamat.train, traindata.y, spcamat.test, Cpar[2])
    svmerrormatB[nm,7] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 7] = length(which((testdata.y-(sign(out)+1)/2)==1))

    spcaout = preProcess(data.frame(x.reducedtsvmC), method = c("pca"),
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindtsvmC] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindtsvmC] %*% spcaout$rotation
    no.featsvmC[nm,7] = snumber.of.comp
    out = Psvm(spcamat.train, traindata.y, spcamat.test, Cpar[3])
    svmerrormatC[nm,7] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 7] = length(which((testdata.y-(sign(out)+1)/2)==1))

##########################################################################
#                                                                       ##
# METHOD 8: Applying KNN & SVM to the SPCs based on corr. thresholding  ##
#                                                                       ##
##########################################################################
    spcaout = preProcess(data.frame(x.reducedcorknnA), method = c("pca"),
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindcorknnA] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindcorknnA] %*% spcaout$rotation
    no.featA[nm,8] = snumber.of.comp
    out = knn(spcamat.train, spcamat.test, as.factor(traindata.y), k = 
Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatA[nm, 8] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm, 8] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    spcaout = preProcess(data.frame(x.reducedcorknnB), method = c("pca"), 
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindcorknnB] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindcorknnB] %*% spcaout$rotation
    no.featB[nm,8] = snumber.of.comp
    out = knn(spcamat.train, spcamat.test, as.factor(traindata.y), k = 
Kvec[2], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatB[nm, 8] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm, 8] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    spcaout = preProcess(data.frame(x.reducedcorknnC), method = c("pca"), 
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindcorknnC] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindcorknnC] %*% spcaout$rotation
    no.featC[nm,8] = snumber.of.comp
    out = knn(spcamat.train, spcamat.test, as.factor(traindata.y), k = 
Kvec[3], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatC[nm, 8] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm, 8] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    spcaout = preProcess(data.frame(x.reducedcorsvmA), method = c("pca"), 
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindcorsvmA] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindcorsvmA] %*% spcaout$rotation
    no.featsvmA[nm,8]= snumber.of.comp
    out = Psvm(spcamat.train, traindata.y, spcamat.test, Cpar[1])
    svmerrormatA[nm,8] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 8] = length(which((testdata.y-(sign(out)+1)/2)==1))


    spcaout = preProcess(data.frame(x.reducedcorsvmB), method = c("pca"), 
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindcorsvmB] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindcorsvmB] %*% spcaout$rotation
    no.featsvmB[nm,8]= snumber.of.comp
    out = Psvm(spcamat.train, traindata.y, spcamat.test, Cpar[2])
    svmerrormatB[nm,8] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 8] = length(which((testdata.y-(sign(out)+1)/2)==1))

    spcaout = preProcess(data.frame(x.reducedcorsvmC), method = c("pca"), 
thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindcorsvmC] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindcorsvmC] %*% spcaout$rotation
    no.featsvmC[nm,8]= snumber.of.comp
    out = Psvm(spcamat.train, traindata.y, spcamat.test, Cpar[3])
    svmerrormatC[nm,8] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 8] = length(which((testdata.y-(sign(out)+1)/2)==1))
```

```
##########################################################################
#                                                                       ##
# METHOD 9: Applying KNN & SVM to the NSC selected variables obtained  ##
#            from Pthreshold_nsc with the optimal nsc.thr value         ##
#                                                                       ##
##########################################################################

  ## OBTAINING REDUCED DATA SET USING NSC PROCEDURE ##
    nsc = Pnsc(traindata.xy,nsc.thr)
    xindnsc = which(apply(nsc$nsc.omit,2,sum)>0)
    x.reducednsc = traindata.x[, xindnsc]
    nscvsmat[nm,xindnsc] = 1

  ## KNN ##
    no.featA[nm,9] =no.featB[nm,9] = no.featC[nm,9]= length(xindnsc)

    out = knn(train=traindata.x[,xindnsc], test=testdata.x[,xindnsc],
as.factor(traindata.y), k = Kvec[1], l = 0, prob = FALSE, use.all =
FALSE)
    knnerrormatA[nm, 9] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm, 9] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(train=traindata.x[,xindnsc], test=testdata.x[,xindnsc],
as.factor(traindata.y), k = Kvec[2], l = 0, prob = FALSE, use.all =
FALSE)
    knnerrormatB[nm, 9] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm, 9] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(train=traindata.x[,xindnsc], test=testdata.x[,xindnsc],
as.factor(traindata.y), k = Kvec[3], l = 0, prob = FALSE, use.all =
FALSE)
    knnerrormatC[nm, 9] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm, 9] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

  ## SVM ##
   no.featsvmA[nm,9]=no.featsvmB[nm,9]=no.featsvmC[nm,9]=length(xindnsc)

    out = Psvm(x.reducednsc, traindata.y, testdata.x[,xindnsc], Cpar[1])
    svmerrormatA[nm,9] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 9] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(x.reducednsc, traindata.y, testdata.x[,xindnsc], Cpar[2])
    svmerrormatB[nm,9] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 9] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(x.reducednsc, traindata.y, testdata.x[,xindnsc], Cpar[3])
    svmerrormatC[nm,9] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 9] = length(which((testdata.y-(sign(out)+1)/2)==1))

##########################################################################
#                                                                       ##
# METHOD 10: Applying LDA to the NSC selected variables obtained from  ##
#             Pthreshold_nsc with the optimal nsc.thr value             ##
#                                                                       ##
##########################################################################

 ## OBTAINING REDUCED DATA SET USING NSC PROCEDURE ##
   nsc = Pnsc(traindata.xy,nsc.thr)
```

```
    xindnsc = which(apply(nsc$nsc.omit,2,sum)>0)
    nscvsmat[nm,xindnsc] = 1
    no.featLDA[nm,1] = length(xindnsc)

   out = PLDA(traindata.x[,xindnsc],traindata.y, testdata.x[,xindnsc])
   LDAerrormat[nm, 1] = sum(as.numeric(out!= testdata.y))
   LDAFNRmat[nm,1] = length(which(sign(testdata.y-out)==1))

#########################################################################
#                                                                      ##
# METHOD 11: Applying the NSC classifier to the NSC selected variables ##
#           obtained from Pthreshold_nsc using the opt. nsc.thr value  ##
#                                                                      ##
#########################################################################
    no.featLDA[nm,2] = length(xindnsc)
    out = Pnsc_class(traindata.x,traindata.y, testdata.x,delta=nsc.thr)
    LDAerrormat[nm, 2] = sum(as.numeric(out!= testdata.y))
    LDAFNRmat[nm,2] = length(which(sign(testdata.y-out)==1))

#########################################################################
#                                                                      ##
# METHOD 12: Applying DLDA using all the input variables in the data   ##
#                                                                      ##
#########################################################################
   no.featLDA[nm,3] = ncol(traindata.x)
   out = PdiagLDA(traindata.x, traindata.y,testdata.x)
   LDAerrormat[nm, 3] = sum(as.numeric(out!= testdata.y))
   LDAFNRmat[nm,3] = length(which(sign(testdata.y-out)==1))

#########################################################################
#                                                                      ##
# METHOD 13: Applying DLDA to the NSC selected variables obtained frm  ##
#            Pthreshold_nsc with the optimal nsc.thr value             ##
#                                                                      ##
#########################################################################
  no.featLDA[nm,4] = length(xindnsc)
  out = PdiagLDA(traindata.x[,xindnsc],traindata.y, testdata.x[,xindnsc])
  LDAerrormat[nm, 4] = sum(as.numeric(out!= testdata.y))
  LDAFNRmat[nm,4] = length(which(sign(testdata.y-out)==1))

  }
#########################################################################
#                                                                      ##
# Returning the output of the error matrix                             ##
#                                                                      ##
#########################################################################

## KNN OUTPUT ##
 out.mat = matrix(rep(0),ncol=4,nrow=9)
 rownames(out.mat)=c("KNN","fastKNN","ttest-KNN","cor-KNN","cor-fastKNN",
                    "PCA-KNN","ttest-SPCA-KNN","cor-SPCA-KNN","NSC-KNN")
 colnames(out.mat)=c("No features","Test error","Std error","FNR")

 out.matA = out.matB= out.matC= out.mat

 knnerror.propA = knnerrormatA/testsize
 knnerror.propB = knnerrormatB/testsize
 knnerror.propC = knnerrormatC/testsize

 knnFNR.propA = knnFNRmatA/testsize
 knnFNR.propB = knnFNRmatB/testsize
 knnFNR.propC = knnFNRmatC/testsize

 out.matA[,1]= apply(no.featA,2,PMode)
 out.matA[,2] = apply(knnerror.propA,2,mean)
 out.matA[,3] = apply(knnerror.propA,2,sd)/sqrt(numsplits)
 out.matA[,4] = apply(knnFNR.propA,2,mean)

 out.matB[,1]= apply(no.featB,2, PMode)
 out.matB[,2] = apply(knnerror.propB,2,mean)
 out.matB[,3] = apply(knnerror.propB,2,sd)/sqrt(numsplits)
 out.matB[,4] = apply(knnFNR.propB,2,mean)

 out.matC[,1]= apply(no.featC,2,PMode)
 out.matC[,2] = apply(knnerror.propC,2,mean)
 out.matC[,3] = apply(knnerror.propC,2,sd)/sqrt(numsplits)
 out.matC[,4] = apply(knnFNR.propC,2,mean)

## SVM OUTPUT ##
 svmout.mat = matrix(rep(0),ncol=4,nrow=9)
 rownames(svmout.mat)=c("SVM","fastKNN-SVM","ttest-SVM","cor-SVM","
","PCA-SVM","ttest-SPCA-SVM","cor-SPCA-SVM","NSC-SVM")
 colnames(svmout.mat)=c("No features","Test error","Std error","FNR")

 svmout.matA = svmout.matB= svmout.matC= svmout.mat

 svmerror.propA = svmerrormatA/testsize
 svmerror.propB = svmerrormatB/testsize
 svmerror.propC = svmerrormatC/testsize

 svmFNR.propA = svmFNRmatA/testsize
 svmFNR.propB = svmFNRmatB/testsize
 svmFNR.propC = svmFNRmatC/testsize

 svmout.matA[,1]= apply(no.featsvmA,2, PMode)
 svmout.matA[,2] = apply(svmerror.propA,2,mean)
 svmout.matA[,3] = apply(svmerror.propA,2,sd)/sqrt(numsplits)
 svmout.matA[,4] = apply(svmFNR.propA,2,mean)

 svmout.matB[,1]= apply(no.featsvmB,2, PMode)
 svmout.matB[,2] = apply(svmerror.propB,2,mean)
 svmout.matB[,3] = apply(svmerror.propB,2,sd)/sqrt(numsplits)
 svmout.matB[,4] = apply(svmFNR.propB,2,mean)

 svmout.matC[,1]= apply(no.featsvmC,2,PMode)
 svmout.matC[,2] = apply(svmerror.propC,2,mean)
 svmout.matC[,3] = apply(svmerror.propC,2,sd)/sqrt(numsplits)
 svmout.matC[,4] = apply(svmFNR.propC,2,mean)

## LDA OUTPUT ##
 LDAout.mat = matrix(rep(0),ncol=4,nrow=4)
 rownames(LDAout.mat)=c("NSC_LDA","NSC","DLDA","NSC_DLDA")
 colnames(LDAout.mat)=c("No features","Test error","Std error","FNR")

 LDAerror.prop = LDAerrormat/testsize
 LDAFNR.prop = LDAFNRmat/testsize

 LDAout.mat[,1]= apply(no.featLDA,2, PMode)
 LDAout.mat[,2] = apply(LDAerror.prop,2,mean)
 LDAout.mat[,3] = apply(LDAerror.prop,2,sd)/sqrt(numsplits)
 LDAout.mat[,4] = apply(LDAFNR.prop,2,mean)
 output = list("KNN.OutputA"=out.matA, "KNN.OutputB"=out.matB,
"KNN.OutputC"=out.matC, "SVM.OutputA"=svmout.matA,
"SVM.OutputB"=svmout.matB,"SVM.OutputC"=svmout.matC,
"LDA.Output" =LDAout.mat,"Ttestvs_featA"=tvsmatA, "Corvs_featA"=rvsmatA,
"Ttestvs_featB"=tvsmatB, "Corvs_featB"=rvsmatB,
"Ttestvs_featC"=tvsmatC, "Corvs_featC"=rvsmatC,
"SVM.Ttestvs_featA"=SVMtvsmatA, "SVM.Corvs_featA"=SVMrvsmatA,
"SVM.Ttestvs_featB"=SVMtvsmatB, "SVM.Corvs_featB"=SVMrvsmatB,
"SVM.Ttestvs_featC"=SVMtvsmatC, "SVM.Corvs_featC"=SVMrvsmatC,
"NSC_vs_feat" =nscvsmat, knnerrormatA, knnerrormatB, knnerrormatC,
svmerrormatA, svmerrormatB, svmerrormatC, LDAerrormat, no.featA,
 no.featB,  no.featC,  no.featsvmA, no.featsvmB, no.featsvmC, no.featLDA)

 return(output)
}
```

B.25 Applying the classification procedures to the Sim2 data set

```
> fix(PMain_sim2)
```

```
function(data.xy,  numsplits, trfrac, Kvec, Cpar, pcathresh, corthr) {
  #Kvec is a vector of 3 values for K in KNN
  #Cpar is a vector of 3 values for the C parameter in RBF SVM
  #corthr is a vector of 6 values

#########################################################################
#                                                                      ##
# This program analyses a given binary data set using different        ##
# versions of the KNN,fastKNN, SVM & DLDA classifier                   ##
#                                                                      ##
# The following procedures are applied:                                ##
# A. KNN/ SVM using all the input variables                            ##
# B. fastkNN/ SVM using features extracted by means of fastKNN         ##
# C. KNN/ SVM using ordinary PCs extracted from the full data set      ##
#                                                                      ##
# D. DLDA using all the input variables                                ##
#                                                                      ##
#########################################################################

#########################################################################
#                                                                      ##
# Function to obtain the mode                                          ##
#                                                                      ##
#########################################################################
 PMode = function(x) {
    ux = unique(x)
    ux[which.max(tabulate(match(x, ux)))]
    }

#########################################################################
#                                                                      ##
# Data properties, Pre-processing the data &                           ##
# Obtain the row indices identifying the Y = 0 & Y = 1 data cases      ##
#                                                                      ##
#########################################################################

 ncol = ncol(data.xy)
 p = ncol-1
 N = nrow(data.xy)
 data.x = as.matrix(data.xy[,-ncol])
 yvec = data.xy[,ncol]
 data.y = yvec
 ind0 = which(yvec == 0)
 ind1 = which(yvec == 1)
 N0 = length(ind0)
 N1 = length(ind1)

 knnerrormatA= knnerrormatB=knnerrormatC=matrix(0,nrow=numsplits,ncol=9)
 knnFNRmatA= knnFNRmatB=knnFNRmatC =matrix(0, nrow =numsplits, ncol =9)

 svmerrormatA =svmerrormatB=svmerrormatC=matrix(0,nrow=numsplits,ncol= 9)
 svmFNRmatA = svmFNRmatB = svmFNRmatC =matrix(0, nrow=numsplits, ncol= 9)

 LDAerrormat = matrix(0, nrow = numsplits, ncol = 4)
 LDAFNRmat = matrix(0, nrow = numsplits, ncol = 4)

 no.featA =matrix(0, nrow=numsplits, ncol=9)
 no.featB = matrix(0, nrow=numsplits, ncol=9)
 no.featC = matrix(0, nrow=numsplits, ncol=9)
 no.featsvmA= no.featsvmB= no.featsvmC= matrix(0, nrow=numsplits, ncol=9)
 no.featLDA = matrix(0, nrow=numsplits, ncol=4)

 trainsize0 = floor(trfrac*N0)
 trainsize1 = floor(trfrac*N1)
 trainsize = trainsize0 + trainsize1
 testsize0 = N0 - trainsize0
 testsize1 = N1 - trainsize1
 testsize = testsize0 + testsize1

 for (nm in 1:numsplits)  {
    print(nm)
    trainind0 = sample (ind0, trainsize0, replace=F)
    trainind1 = sample (ind1, trainsize1, replace=F)
    trainind = c(trainind0,trainind1)
    traindata.x = data.x[trainind,]
    traindata.y = data.y[trainind]
    traindata.xy = cbind(traindata.x, traindata.y)
    testdata.x = data.x[-trainind,]
    testdata.y = data.y[-trainind]

##########################################################################
#                                                                       ##
# METHOD A: Applying ordinary kNN/ SVM using all the input variables    ##
#                                                                       ##
##########################################################################

 no.featA[nm,1] =no.featB[nm,1]=no.featC[nm,1]= ncol(traindata.x)

    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatA[nm, 1] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,1] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[2], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatB[nm, 1] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,1] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))
```

```
    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[3], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatC[nm, 1] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,1] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

##Note that in SVM the sign(fvalues) = -1 or 1 & need to be transformed##

no.featsvmA[nm,1]=no.featsvmB[nm,1]=no.featsvmC[nm,1]=ncol(traindata.x)

    out = Psvm(traindata.x, traindata.y, testdata.x,Cpar[1])
    svmerrormatA[nm, 1] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 1] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(traindata.x,traindata.y,testdata.x,Cpar[2])
    svmerrormatB[nm,1] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 1] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(traindata.x,traindata.y,testdata.x,Cpar[3])
    svmerrormatC[nm,1] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 1] = length(which((testdata.y-(sign(out)+1)/2)==1))

#########################################################################
#                                                                      ##
# METHOD B: Applying the fastKNN classifier to the fastKNN extracted   ##
#           Applying SVM classifier to fastKNN features using K=Kvec[3]##
#                                                                      ##
#########################################################################

    extracted.features =PfastKNNset (traindata.x,traindata.y,Kvec[1],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featA[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[1])$class
    knnerrormatA[nm, 2] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,2] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    extracted.features =PfastKNNset (traindata.x,traindata.y,Kvec[2],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featB[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[2])$class
    knnerrormatB[nm, 2] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,2] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    extracted.features =PfastKNNset (traindata.x,traindata.y,Kvec[3],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featC[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[3])$class
    knnerrormatC[nm, 2] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,2] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))
```

```
 no.featsvmA[nm,2]=no.featsvmB[nm,2]=no.featsvmC[nm,2] = ncol(extr.train)
    out = Psvm(extr.train,traindata.y, extr.test,Cpar[1])
    svmerrormatA[nm,2] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 2] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(extr.train,traindata.y, extr.test,Cpar[2])
    svmerrormatB[nm,2] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 2] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(extr.train,traindata.y, extr.test,Cpar[3])
    svmerrormatC[nm,2] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 2] = length(which((testdata.y-(sign(out)+1)/2)==1))

##########################################################################
#                                                                       ##
# METHOD C: Applying KNN & SVM to the PC extracted from the full data   ##
#                                                                       ##
##########################################################################
pcaout =preProcess(data.frame(traindata.x),method=c("pca"),thresh=
pcathresh)
    number.of.comp = pcaout$numComp
    pcamat.train = traindata.x%*%pcaout$rotation
    pcamat.test = testdata.x%*%pcaout$rotation

   no.featA[nm,3]=no.featB[nm,3]=no.featC[nm,3] = number.of.comp

    out = knn(pcamat.train, pcamat.test, as.factor(traindata.y), k =
Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatA[nm, 3] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,3] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(pcamat.train, pcamat.test, as.factor(traindata.y), k =
Kvec[2], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatB[nm, 3] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,3] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(pcamat.train, pcamat.test, as.factor(traindata.y), k =
Kvec[3], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatC[nm, 3] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,3] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    no.featsvmA[nm,3] =no.featsvmB[nm,3]=no.featsvmC[nm,3]=number.of.comp

    out = Psvm(pcamat.train, traindata.y, pcamat.test, Cpar[1])
    svmerrormatA[nm,3] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 3] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(pcamat.train, traindata.y, pcamat.test, Cpar[2])
    svmerrormatB[nm,3] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 3] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(pcamat.train, traindata.y, pcamat.test, Cpar[3])
    svmerrormatC[nm,3] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 3] = length(which((testdata.y-(sign(out)+1)/2)==1))
```

```
##########################################################################
#                                                                       ##
# METHOD D: Applying DLDA using all the input variables in the data    ##
#                                                                       ##
##########################################################################
   no.featLDA[nm,1] = ncol(traindata.x)
   out = PdiagLDA(traindata.x, traindata.y,testdata.x)
   LDAerrormat[nm, 1] = sum(as.numeric(out!= testdata.y))
   LDAFNRmat[nm,1] = length(which(sign(testdata.y-out)==1))
 }

##########################################################################
#                                                                       ##
# Returning the output of the error matrix                              ##
#                                                                       ##
##########################################################################
## KNN OUTPUT ####
 out.mat = matrix(rep(0),ncol=4,nrow=3)
 rownames(out.mat)=c("KNN","fastKNN","PCA-KNN")
 colnames(out.mat)=c("No features","Test error","Std error","FNR")

 out.matA = out.matB= out.matC= out.mat

 knnerror.propA = knnerrormatA/testsize
 knnerror.propB = knnerrormatB/testsize
 knnerror.propC = knnerrormatC/testsize

 knnFNR.propA = knnFNRmatA/testsize
 knnFNR.propB = knnFNRmatB/testsize
 knnFNR.propC = knnFNRmatC/testsize

 out.matA[,1]= apply(no.featA,2,PMode)
 out.matA[,2] = apply(knnerror.propA,2,mean)
 out.matA[,3] = apply(knnerror.propA,2,sd)/sqrt(numsplits)
 out.matA[,4] = apply(knnFNR.propA,2,mean)

 out.matB[,1]= apply(no.featB,2, PMode)
 out.matB[,2] = apply(knnerror.propB,2,mean)
 out.matB[,3] = apply(knnerror.propB,2,sd)/sqrt(numsplits)
 out.matB[,4] = apply(knnFNR.propB,2,mean)

 out.matC[,1]= apply(no.featC,2,PMode)
 out.matC[,2] = apply(knnerror.propC,2,mean)
 out.matC[,3] = apply(knnerror.propC,2,sd)/sqrt(numsplits)
 out.matC[,4] = apply(knnFNR.propC,2,mean)

## SVM OUTPUT ##
 svmout.mat = matrix(rep(0),ncol=4,nrow=3)
 rownames(svmout.mat)=c("SVM","fastKNN-SVM","PCA-SVM")
 colnames(svmout.mat)=c("No features","Test error","Std error","FNR")

 svmout.matA = svmout.matB= svmout.matC= svmout.mat

 svmerror.propA = svmerrormatA/testsize
 svmerror.propB = svmerrormatB/testsize
 svmerror.propC = svmerrormatC/testsize

 svmFNR.propA = svmFNRmatA/testsize
 svmFNR.propB = svmFNRmatB/testsize
 svmFNR.propC = svmFNRmatC/testsize
 svmout.matA[,1]= apply(no.featsvmA,2, PMode)
 svmout.matA[,2] = apply(svmerror.propA,2,mean)
 svmout.matA[,3] = apply(svmerror.propA,2,sd)/sqrt(numsplits)
 svmout.matA[,4] = apply(svmFNR.propA,2,mean)

 svmout.matB[,1]= apply(no.featsvmB,2, PMode)
 svmout.matB[,2] = apply(svmerror.propB,2,mean)
 svmout.matB[,3] = apply(svmerror.propB,2,sd)/sqrt(numsplits)
 svmout.matB[,4] = apply(svmFNR.propB,2,mean)

 svmout.matC[,1]= apply(no.featsvmC,2,PMode)
 svmout.matC[,2] = apply(svmerror.propC,2,mean)
 svmout.matC[,3] = apply(svmerror.propC,2,sd)/sqrt(numsplits)
 svmout.matC[,4] = apply(svmFNR.propC,2,mean)

## LDA OUTPUT ##
 LDAout.mat = matrix(rep(0),ncol=4,nrow=1)
 rownames(LDAout.mat)=c("DLDA")
 colnames(LDAout.mat)=c("No features","Test error","Std error","FNR")

 LDAerror.prop = LDAerrormat/testsize
 LDAFNR.prop = LDAFNRmat/testsize

 LDAout.mat[,1]= apply(no.featLDA,2, PMode)
 LDAout.mat[,2] = apply(LDAerror.prop,2,mean)
 LDAout.mat[,3] = apply(LDAerror.prop,2,sd)/sqrt(numsplits)
 LDAout.mat[,4] = apply(LDAFNR.prop,2,mean)


 output = list("KNN.OutputA"=out.matA, "KNN.OutputB"=out.matB,
    "KNN.OutputC"=out.matC,"SVM.OutputA"=svmout.matA,
    "SVM.OutputB"=svmout.matB, "SVM.OutputC"=svmout.matC,
    "LDA.Output" =LDAout.mat,knnerrormatA, knnerrormatB, knnerrormatC,
    svmerrormatA, svmerrormatB, svmerrormatC, LDAerrormat,no.featA ,
    no.featB,  no.featC,  no.featsvmA, no.featsvmB, no.featsvmC)

 return(output)
}
```

B.26 Applying the classification procedures to the multi-class data set

```
>  fix(PMain_mult)
```

```
function(data.xy, Kvec, numsplits, trfrac, pcathresh, cor.thr, nsc.thr) {
  # Kvec is a vector of 3 values for K in KNN
  # cor.thr is a vector of 3 value
  # nsc.thr is a vector of 1 values  (optimal value from NSC classifier)

#########################################################################
#                                                                     ##
# This program analyses a given multi-class data set using different  ##
# versions of the KNN,fastKNN, & NSC classifier                       ##
#                                                                     ##
# The following procedures are applied:                               ##
# 1. KNN using all the input variables                                ##
# 2. fastkNN using features extracted by means of fastKNN package     ##
# 3. KNN using only variables selected by correlation thresholding    ##
# 4. fastkNN using features extracted based only on correlation thresh.##
# 5. KNN using ordinary PCs extracted from the full data set          ##
# 6. KNN using supervised PCs based on correlation thresholding       ##
# 7. KNN using NSC selected features from opt. delta threshlold        ##
#                                                                      ##
# 8. NSC using NSC select features from opt. delta threshold           ##
#                                                                      ##
#########################################################################
#########################################################################

#########################################################################
#                                                                      ##
# Function to obtain the mode                                          ##
#                                                                      ##
#########################################################################
 PMode = function(x) {
    ux = unique(x)
    ux[which.max(tabulate(match(x, ux)))]
    }

#########################################################################
#                                                                      ##
# Data properties, Pre-processing the data &                           ##
# Obtain the row indices identifying the row indicies for Y =1,2,3,4   ##
#                                                                      ##
#########################################################################

 ncol = ncol(data.xy)
 p = ncol-1
 N = nrow(data.xy)
 data.x = as.matrix(data.xy[,-ncol])
 yvec = data.xy[,ncol]
 data.y = yvec
 G = length(unique(yvec))        #no of classes
 Ng = rep(0,G)
 Ngtrain = rep(0,G)
 Ngtest = rep(0,G)

 indg = matrix(rep(0),ncol=G,nrow=N)

 for (g in 1:G) {
  indg[,g] = ifelse(yvec == g,1,0)
  Ng[g] = sum(indg[,g])
  Ngtrain[g] = floor(trfrac*Ng[g])
  Ngtest[g] = Ng[g] - Ngtrain[g]
  }

 errormatA = errormatB =errormatC = matrix(0, nrow = numsplits, ncol = 7)
 no.featA =no.featB =no.featC  = matrix(0, nrow = numsplits, ncol = 7)
 errormat =  matrix(0, nrow = numsplits, ncol = 1)
 no.feat =matrix(0, nrow = numsplits, ncol = 1)
 trainsize = sum(Ngtrain)
 testsize = sum(Ngtest)

 # Variables indices selected by ##
 rvsmatA =rvsmatB =rvsmatC = matrix(0, nrow=numsplits, ncol=p) #CORR
 nscvsmat =nscvsmatKNN = matrix(0, nrow=numsplits, ncol=p)   # NSC


  for (nm in 1:numsplits)  {
   print(nm)
   trainind = NULL
   for (g in 1:G)
   trainind=c(trainind,sample(which(indg[,g]>0),Ngtrain[g],replace=F))
```

```
    traindata.x = data.x[trainind,]
    traindata.y = data.y[trainind]
    traindata.xy = cbind(traindata.x, traindata.y)
    testdata.x = data.x[-trainind,]
    testdata.y = data.y[-trainind]

########################################################################
#                                                                     ##
# METHOD 1: Applying ordinary kNN to all input variables in the data  ##
#                                                                     ##
########################################################################

   no.featA[nm,1] = ncol(traindata.x)
    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    errormatA[nm, 1] = sum(as.numeric(out) != testdata.y)

    no.featB[nm,1] = ncol(traindata.x)
    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[2], l = 0, prob = FALSE, use.all = FALSE)
    errormatB[nm, 1] = sum(as.numeric(out) != testdata.y)

    no.featC[nm,1] = ncol(traindata.x)
    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[3], l = 0, prob = FALSE, use.all = FALSE)
    errormatC[nm, 1] = sum(as.numeric(out) != testdata.y)

########################################################################
#                                                                     ##
# METHOD 2: Applying the fastKNN classifier to the fastKNN features   ##
#                                                                     ##
########################################################################

   extracted.features =PfastKNNset_mult(traindata.x,traindata.y,Kvec[1],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featA[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[1])$class
    errormatA[nm, 2] = sum(as.numeric(out) != testdata.y)

    extracted.features =PfastKNNset_mult(traindata.x,traindata.y,Kvec[2],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featB[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[2])$class
    errormatB[nm, 2] = sum(as.numeric(out) != testdata.y)

    extracted.features =PfastKNNset_mult(traindata.x,traindata.y,Kvec[3],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featC[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[3])$class
    errormatC[nm, 2] = sum(as.numeric(out) != testdata.y)
```

```
########################################################################
#                                                                    ##
# METHOD 3: Applying KNN to the correlation thresholding selected vars ##
#                                                                    ##
########################################################################
   corout = Pcor(traindata.xy, cor.thr[1])
    xindcorA= corout$Indices
    rvsmatA[nm,xindcorA] = 1
    no.featA[nm,3] = length(xindcorA)
    out = knn(traindata.x[,xindcorA], testdata.x[,xindcorA], k= Kvec[1],
          as.factor(traindata.y), l= 0, prob = FALSE, use.all = FALSE)
    errormatA[nm, 3] = sum((as.numeric(out)) != testdata.y)

    corout = Pcor(traindata.xy, cor.thr[2])
    xindcorB= corout$Indices
    rvsmatB[nm,xindcorB] = 1
    no.featB[nm,3] = length(xindcorB)
    out = knn(traindata.x[,xindcorB], testdata.x[,xindcorB], k = Kvec[2],
          as.factor(traindata.y), l = 0, prob = FALSE, use.all = FALSE)
    errormatB[nm, 3] = sum((as.numeric(out)) != testdata.y)

    corout = Pcor(traindata.xy, cor.thr[3])
    xindcorC= corout$Indices
    rvsmatC[nm,xindcorC] = 1
    no.featC[nm,3] = length(xindcorC)
    out = knn(traindata.x[,xindcorC], testdata.x[,xindcorC], k = Kvec[3],
         as.factor(traindata.y), l = 0, prob = FALSE, use.all = FALSE)
    errormatC[nm, 3] = sum((as.numeric(out)) != testdata.y)

########################################################################
#                                                                    ##
# METHOD 4: Applying the fastKNN classifier using the fastKNN features ##
#           extracted from the correlation thresholded data set        ##
#                                                                    ##
########################################################################
    extracted.features =PfastKNNset_mult
(traindata.x[,xindcorA],traindata.y,Kvec[1], testdata.x[,xindcorA])
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featA[nm,4] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[1])$class
    errormatA[nm, 4] = sum((as.numeric(out)) != testdata.y)

    extracted.features =PfastKNNset_mult
(traindata.x[,xindcorB],traindata.y,Kvec[2], testdata.x[,xindcorB])
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featB[nm,4] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[2])$class
    errormatB[nm, 4] = sum((as.numeric(out)) != testdata.y)

    extracted.features =PfastKNNset_mult
(traindata.x[,xindcorC],traindata.y,Kvec[3], testdata.x[,xindcorC])
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featC[nm,4] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[3])$class
    errormatC[nm, 4] = sum((as.numeric(out)) != testdata.y)
```

```
##########################################################################
#                                                                       ##
# METHOD 5: Applying KNN to the PC extracted from the full data        ##
#                                                                       ##
##########################################################################

    pcaout = preProcess(data.frame(traindata.x), method = c("pca"),
thresh = pcathresh)
    number.of.comp = pcaout$numComp
    pcamat.train = traindata.x%*%pcaout$rotation
    pcamat.test = testdata.x%*%pcaout$rotation

    no.featA[nm,5]= number.of.comp
    out = knn(pcamat.train, pcamat.test, as.factor(traindata.y), k =
Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    errormatA[nm, 5] = sum((as.numeric(out)) != testdata.y)

    no.featB[nm,5]= number.of.comp
    out = knn(pcamat.train, pcamat.test, as.factor(traindata.y), k =
Kvec[2], l = 0, prob = FALSE, use.all = FALSE)
    errormatB[nm, 5] = sum((as.numeric(out)) != testdata.y)

    no.featC[nm,5]= number.of.comp
    out = knn(pcamat.train, pcamat.test, as.factor(traindata.y), k =
Kvec[3], l = 0, prob = FALSE, use.all = FALSE)
    errormatC[nm, 5] = sum((as.numeric(out)) != testdata.y)

##########################################################################
#                                                                       ##
# METHOD 6: Applying KNN to the SPCs based on correlation thresholding ##
#                                                                       ##
##########################################################################

    spcaout = preProcess(data.frame(traindata.x[,xindcorA]), method =
c("pca"), thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindcorA] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindcorA] %*% spcaout$rotation
    no.featA[nm,6] = snumber.of.comp
    out = knn(spcamat.train, spcamat.test, as.factor(traindata.y), k =
Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    errormatA[nm, 6] = sum((as.numeric(out)) != testdata.y)

    spcaout = preProcess(data.frame(traindata.x[,xindcorB]), method =
c("pca"), thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindcorB] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindcorB] %*% spcaout$rotation
    no.featB[nm,6] = snumber.of.comp
    out = knn(spcamat.train, spcamat.test, as.factor(traindata.y), k =
Kvec[2], l = 0, prob = FALSE, use.all = FALSE)
    errormatB[nm, 6] = sum((as.numeric(out)) != testdata.y)

    spcaout = preProcess(data.frame(traindata.x[,xindcorC]), method =
c("pca"), thresh = pcathresh)
    snumber.of.comp = spcaout$numComp
    spcamat.train = traindata.x[, xindcorC] %*% spcaout$rotation
    spcamat.test = testdata.x[, xindcorC] %*% spcaout$rotation
    no.featC[nm,6] = snumber.of.comp
    out = knn(spcamat.train, spcamat.test, as.factor(traindata.y), k =
Kvec[3], l = 0, prob = FALSE, use.all = FALSE)
```

```
    errormatC[nm, 6] = sum((as.numeric(out)) != testdata.y)

##########################################################################
#                                                                       ##
# METHOD 7: Applying the KNN classifier to the NSC selected variables   ##
#           obtained from Pthreshold_nsc using the opt. nsc.thr value   ##
#                                                                       ##
##########################################################################

    nsc = Pnsc_mult(traindata.xy,nsc.thr)
    xindnsc = which(apply(nsc$nsc.omit,2,sum)>0)
    nscvsmatKNN[nm,xindnsc] = 1

    no.featA[nm,7] = length(xindnsc)
    out = knn(train=traindata.x[,xindnsc], test=testdata.x[,xindnsc],
as.factor(traindata.y), k = Kvec[1], l =0, prob = FALSE, use.all = FALSE)
    errormatA[nm, 7] = sum((as.numeric(out)) != testdata.y)

    no.featB[nm,7] = length(xindnsc)
    out = knn(train=traindata.x[,xindnsc], test=testdata.x[,xindnsc],
as.factor(traindata.y), k = Kvec[2], l =0, prob = FALSE, use.all = FALSE)
    errormatB[nm, 7] = sum((as.numeric(out)) != testdata.y)


    no.featC[nm,7] = length(xindnsc)
    out = knn(train=traindata.x[,xindnsc], test=testdata.x[,xindnsc],
as.factor(traindata.y), k = Kvec[3], l =0, prob = FALSE, use.all = FALSE)
    errormatC[nm, 7] = sum((as.numeric(out)) != testdata.y)
##########################################################################
#                                                                       ##
# METHOD 8: Applying the NSC classifier to the NSC selected variables   ##
#           obtained from Pthreshold_nsc using the opt. nsc.thr value   ##
#                                                                       ##
##########################################################################
 nscvsmat[nm,xindnsc] = 1
 no.feat[nm,1] = length(xindnsc)

  out =Pnsc_class_mult(traindata.x,traindata.y, testdata.x,delta=nsc.thr)
    errormat[nm, 1] = sum(as.numeric(out!= testdata.y))
 }

##########################################################################
#                                                                       ##
# Returning the output of the error matrix                              ##
#                                                                       ##
##########################################################################

 out.mat = matrix(rep(0),ncol=3,nrow=7)
 rownames(out.mat)=c("KNN","fastKNN","cor-KNN","cor-fastKNN","PCA-
KNN","cor-SPCA-KNN","NSC-KNN")
 colnames(out.mat)=c("No features","Test error","Std error")

 out.matA = out.matB= out.matC= out.mat

 error.propA = errormatA/testsize
 error.propB = errormatB/testsize
 error.propC = errormatC/testsize

 out.matA[,1]= apply(no.featA,2,PMode)
 out.matA[,2] = apply(error.propA,2,mean)
 out.matA[,3] = apply(error.propA,2,sd)/sqrt(numsplits)
 out.matB[,1]= apply(no.featB,2, PMode)
 out.matB[,2] = apply(error.propB,2,mean)
 out.matB[,3] = apply(error.propB,2,sd)/sqrt(numsplits)

 out.matC[,1]= apply(no.featC,2,PMode)
 out.matC[,2] = apply(error.propC,2,mean)
 out.matC[,3] = apply(error.propC,2,sd)/sqrt(numsplits)

 NSC.mat = matrix(rep(0),ncol=3,nrow=1)
 colnames(NSC.mat)=c("No features","Test error","Std error")
 error.prop = errormat/testsize
 NSC.mat[,1]= apply(no.feat,2,PMode)
 NSC.mat[,2] = apply(error.prop,2,mean)
 NSC.mat[,3] = apply(error.prop,2,sd)/sqrt(numsplits)


 output = list("KNN A mat"=out.matA,"KNN B mat"=out.matB,
               "KNN C mat"=out.matC,"NSC mat"=NSC.mat,
               rvsmatA,rvsmatB,rvsmatC, nscvsmatKNN, nscvsmat,
               errormatA, errormatB, errormatC, errormat,
               no.featA, no.featB, no.featC,no.feat)

 return(output)
}
```

B.27 Applying the classification procedures to p_{rel} in Sim data sets

```
> fix(PMain_sim.rel)
```

```
function(data.xy, numsplits, trfrac,prel, Kvec, Cpar) {
  #Kvec is a vector of 3 values for K in KNN
  #Cpar is a vector of 3 values for the C parameter in RBF SVM
  #prel number of relevant variables to retain in data.xy

##########################################################################
#                                                                       ##
# This program analyses a given binary data set using different         ##
# versions of the KNN,fastKNN, SVM, LDA, DLDA & NSC classifier          ##
#                                                                       ##
# The following procedures are applied:                                 ##
# 1. KNN/ SVM using all the prel input variables                        ##
# 2. fastkNN/ SVM using features extracted by means of fastKNN          ##
#                                                                       ##
# 12. DLDA using all the prel input variables                           ##
#                                                                       ##
##########################################################################


##########################################################################
#                                                                       ##
# Function to obtain the mode                                           ##
#                                                                       ##
##########################################################################
 PMode = function(x) {
    ux = unique(x)
    ux[which.max(tabulate(match(x, ux)))]
    }
```

```
#########################################################################
#                                                                      ##
# Data properties, Pre-processing the data &                           ##
# Obtain the row indices identifying the Y = 0 & Y = 1 data cases      ##
#                                                                      ##
#########################################################################

 ncol = ncol(data.xy)
 data.xy = data.xy[,c(1:p.rel,ncol)]
 ncol = ncol(data.xy)
 p = ncol-1
 N = nrow(data.xy)
 data.x = as.matrix(data.xy[,-ncol])
 yvec = data.xy[,ncol]
 data.y = yvec
 ind0 = which(yvec == 0)
 ind1 = which(yvec == 1)
 N0 = length(ind0)
 N1 = length(ind1)

 knnerrormatA= knnerrormatB=knnerrormatC=matrix(0,nrow=numsplits,ncol=2)
 knnFNRmatA= knnFNRmatB=knnFNRmatC =matrix(0, nrow =numsplits, ncol =2)

 svmerrormatA =svmerrormatB=svmerrormatC=matrix(0,nrow=numsplits,ncol= 2)
 svmFNRmatA = svmFNRmatB = svmFNRmatC =matrix(0, nrow=numsplits, ncol= 2)

 LDAerrormat = matrix(0, nrow = numsplits, ncol = 1)
 LDAFNRmat = matrix(0, nrow = numsplits, ncol = 1)

 no.featA =matrix(0, nrow=numsplits, ncol=2)
 no.featB = matrix(0, nrow=numsplits, ncol=2)
 no.featC = matrix(0, nrow=numsplits, ncol=2)
 no.featsvmA= no.featsvmB= no.featsvmC= matrix(0, nrow=numsplits, ncol=2)
 no.featLDA = matrix(0, nrow=numsplits, ncol=1)

 trainsize0 = floor(trfrac*N0)
 trainsize1 = floor(trfrac*N1)
 trainsize = trainsize0 + trainsize1
 testsize0 = N0 - trainsize0
 testsize1 = N1 - trainsize1
 testsize = testsize0 + testsize1

 for (nm in 1:numsplits)  {
    print(nm)
    trainind0 = sample (ind0, trainsize0, replace=F)
    trainind1 = sample (ind1, trainsize1, replace=F)
    trainind = c(trainind0,trainind1)
    traindata.x = data.x[trainind,]
    traindata.y = data.y[trainind]
    traindata.xy = cbind(traindata.x, traindata.y)
    testdata.x = data.x[-trainind,]
    testdata.y = data.y[-trainind]

#########################################################################
#                                                                      ##
# METHOD 1: Applying ordinary kNN/ SVM using all the input variables   ##
#                                                                      ##
#########################################################################


  no.featA[nm,1] =no.featB[nm,1]=no.featC[nm,1]= ncol(traindata.x)
```

```
    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[1], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatA[nm, 1] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,1] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[2], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatB[nm, 1] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,1] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    out = knn(traindata.x, testdata.x, as.factor(traindata.y), k =
Kvec[3], l = 0, prob = FALSE, use.all = FALSE)
    knnerrormatC[nm, 1] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,1] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

##Note that in SVM the sign(fvalues) = -1 or 1 & need to be transformed##

no.featsvmA[nm,1]=no.featsvmB[nm,1]=no.featsvmC[nm,1]=ncol(traindata.x)

    out = Psvm(traindata.x, traindata.y, testdata.x,Cpar[1])
    svmerrormatA[nm, 1] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 1] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(traindata.x,traindata.y,testdata.x,Cpar[2])
    svmerrormatB[nm,1] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 1] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(traindata.x,traindata.y,testdata.x,Cpar[3])
    svmerrormatC[nm,1] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 1] = length(which((testdata.y-(sign(out)+1)/2)==1))

########################################################################
#                                                                     ##
# METHOD 2: Applying the fastKNN classifier to the fastKNN extracted  ##
#           Applying SVM classifier to fastKNN features using K=Kvec[3]##
#                                                                     ##
########################################################################

    extracted.features =PfastKNNset (traindata.x,traindata.y,Kvec[1],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featA[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[1])$class
    knnerrormatA[nm, 2] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatA[nm,2] = length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    extracted.features =PfastKNNset (traindata.x,traindata.y,Kvec[2],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featB[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[2])$class
    knnerrormatB[nm, 2] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatB[nm,2] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

    extracted.features =PfastKNNset (traindata.x,traindata.y,Kvec[3],
testdata.x)
    extr.train = extracted.features$new.feat.tr
    extr.test = extracted.features$new.feat.te
    no.featC[nm,2] = ncol(extr.train)
    out = fastknn(extr.train, as.factor(traindata.y), extr.test, k =
Kvec[3])$class
    knnerrormatC[nm, 2] = sum((as.numeric(out)-1) != testdata.y)
    knnFNRmatC[nm,2] =length(which(sign(testdata.y-(as.numeric(out)-
1))==1))

 no.featsvmA[nm,2]=no.featsvmB[nm,2]=no.featsvmC[nm,2] = ncol(extr.train)
    out = Psvm(extr.train,traindata.y, extr.test,Cpar[1])
    svmerrormatA[nm,2] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatA[nm, 2] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(extr.train,traindata.y, extr.test,Cpar[2])
    svmerrormatB[nm,2] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatB[nm, 2] = length(which((testdata.y-(sign(out)+1)/2)==1))

    out = Psvm(extr.train,traindata.y, extr.test,Cpar[3])
    svmerrormatC[nm,2] = length(which((sign(out)+1)/2 != testdata.y))
    svmFNRmatC[nm, 2] = length(which((testdata.y-(sign(out)+1)/2)==1))

##########################################################################
#                                                                       ##
# METHOD 12: Applying DLDA using all the input variables in the data    ##
#                                                                       ##
##########################################################################
  no.featLDA[nm,1] = ncol(traindata.x)
  out = PdiagLDA(traindata.x, traindata.y,testdata.x)
  LDAerrormat[nm, 1] = sum(as.numeric(out!= testdata.y))
  LDAFNRmat[nm,1] = length(which(sign(testdata.y-out)==1))

}
##########################################################################
#                                                                       ##
# Returning the output of the error matrix                              ##
#                                                                       ##
##########################################################################

## KNN OUTPUT ##
 out.mat = matrix(rep(0),ncol=3,nrow=2)
 rownames(out.mat)=c("KNN","fastKNN")
 colnames(out.mat)=c("No features","Test error","Std error")

 out.matA = out.matB= out.matC= out.mat

 knnerror.propA = knnerrormatA/testsize
 knnerror.propB = knnerrormatB/testsize
 knnerror.propC = knnerrormatC/testsize

 out.matA[,1]= apply(no.featA,2,PMode)
 out.matA[,2] = apply(knnerror.propA,2,mean)
 out.matA[,3] = apply(knnerror.propA,2,sd)/sqrt(numsplits)

 out.matB[,1]= apply(no.featB,2, PMode)
 out.matB[,2] = apply(knnerror.propB,2,mean)
 out.matB[,3] = apply(knnerror.propB,2,sd)/sqrt(numsplits)

 out.matC[,1]= apply(no.featC,2,PMode)
 out.matC[,2] = apply(knnerror.propC,2,mean)
 out.matC[,3] = apply(knnerror.propC,2,sd)/sqrt(numsplits)

## SVM OUTPUT ##
 svmout.mat = matrix(rep(0),ncol=3,nrow=2)
 rownames(svmout.mat)=c("SVM","fastKNN-SVM")
 colnames(svmout.mat)=c("No features","Test error","Std error")

 svmout.matA = svmout.matB= svmout.matC= svmout.mat

 svmerror.propA = svmerrormatA/testsize
 svmerror.propB = svmerrormatB/testsize
 svmerror.propC = svmerrormatC/testsize

 svmout.matA[,1]= apply(no.featsvmA,2, PMode)
 svmout.matA[,2] = apply(svmerror.propA,2,mean)
 svmout.matA[,3] = apply(svmerror.propA,2,sd)/sqrt(numsplits)

 svmout.matB[,1]= apply(no.featsvmB,2, PMode)
 svmout.matB[,2] = apply(svmerror.propB,2,mean)
 svmout.matB[,3] = apply(svmerror.propB,2,sd)/sqrt(numsplits)

 svmout.matC[,1]= apply(no.featsvmC,2,PMode)
 svmout.matC[,2] = apply(svmerror.propC,2,mean)
 svmout.matC[,3] = apply(svmerror.propC,2,sd)/sqrt(numsplits)

## LDA OUTPUT ##
 LDAout.mat = matrix(rep(0),ncol=3,nrow=1)
 rownames(LDAout.mat)=c("DLDA")
 colnames(LDAout.mat)=c("No features","Test error","Std error")

 LDAerror.prop = LDAerrormat/testsize

 LDAout.mat[,1]= apply(no.featLDA,2, PMode)
 LDAout.mat[,2] = apply(LDAerror.prop,2,mean)
 LDAout.mat[,3] = apply(LDAerror.prop,2,sd)/sqrt(numsplits)

 output = list("KNN.OutputA"=out.matA, "KNN.OutputB"=out.matB,
  "KNN.OutputC"=out.matC, "SVM.OutputA"=svmout.matA,
  "SVM.OutputB"=svmout.matB,"SVM.OutputC"=svmout.matC,
  "LDA.Output" =LDAout.mat,knnerrormatA, knnerrormatB, knnerrormatC,
  svmerrormatA, svmerrormatB, svmerrormatC, LDAerrormat, no.featA,
  no.featB,  no.featC, no.featsvmA, no.featsvmB, no.featsvmC, no.featLDA)

 return(output)
}
```

B.28 Plotting the results PMain for the binary data sets

```
> fix(PFigure_main)
```

```
function(data.xy, knn_out, svm_out) {
  # knn_out output from the PMain on binary data for optimal Kvec value
  # svm_out output from the PMain on binary data for optimal C value

#########################################################################
#                                                                     ##
# This program plots a graph based on the output from PMain_          ##
#                                                                     ##
#########################################################################

 ## Getting details for the plot skeleton ##
  data.x = as.matrix(data.xy[,-ncol(data.xy)])
  yvec = data.xy[,ncol(data.xy)]
  p = ncol(data.x)
  N = nrow(data.xy)
  ind0 = which(yvec == 0)
  ind1 = which(yvec == 1)
  N0 = length(ind0)
  N1 = length(ind1)

#########################################################################
#                                                                     ##
# Plotting the basic details for the data set                 _       ##
#                                                                     ##
#########################################################################
 plot(x=0, y=0, ty="n", xlim=c(0,p), ylim = c(0,0.5), ylab="Test Error",
      xlab="Number of Features used")

## Plotting vertical line at p=N ##
   abline(v=N, lty="dashed")
   mtext(text="N",side=1,at=N)            #labelling p=N line

## Plotting random guessing line ##
   abline(h=0.5, lty="dashed")
   mtext(text="0.5",side=2,at=0.5)         #labelling random guessing
line

## Plotting the majority vote ##
   majority=min(N1,N0)/N
   abline(h=majority, lty="dotted")
   mtext(text="Majority",side=2,at=majority)  #labelling majority vote

#########################################################################
#                                                                     ##
# Plotting the KNN output for PMain for the optimal Kvec value        ##
#                                                                     ##
#########################################################################
  knn.feat = knn_out[,1]
  knn.error = knn_out[,2]
  knn.stderr = knn_out[,3]


  cols = c("red","chocolate","yellowgreen","green3","turquoise","blue",
        "royalblue","purple","magenta")
  points(x=knn.feat, y=knn.error, pch=22, col=cols, lwd=knn.stderr*5000)

  ## KNN Legend ##
  names = rownames(knn_out)
  legend("topright", legend=names, col= cols, pch=22)

#########################################################################
#                                                                      ##
# Plotting the SVM output for PMain for the optimal Cpar value         ##
#                                                                      ##
#########################################################################
  svm.feat = svm_out[-5,1]
  svm.error = svm_out[-5,2]
  svm.stderr = svm_out[-5,3]

  colsvm = c("red","chocolate","yellowgreen","green3","blue","royalblue",
             "purple","magenta")
  points(x=svm.feat, y=svm.error,pch=19,col=colsvm, lwd=svm.stderr*5000)

 ## SVM Legend ##
 namesvm = rownames(svm_out)
 legend("topright", legend=namesvm, col= colsvm, pch=19)

}
```

B.29 Plotting the results of the multi-class data set

```
> fix(PFigure_main_mult)
```

```
function(data.xy, knn_out, nsc_out) {
  # knn_out output obtained from PMain_mult for optimal Kvec value
  # nsc_out output obtained from PMain_mult for NSC classifier

#########################################################################
#                                                                      ##
# This program plots a graph based on the output from PMain_mult       ##
#                                                                      ##
#########################################################################

 ## Getting details for the plot skeleton ##
 data.x = as.matrix(data.xy[,-ncol(data.xy)])
 yvec = data.xy[,ncol(data.xy)]
 p = ncol(data.x)
 N = nrow(data.xy)
 ind1 = which(yvec == 1);  N1 = length(ind1)
 ind2 = which(yvec == 2);  N2 = length(ind2)
 ind3 = which(yvec == 3);  N3 = length(ind3)
 ind4 = which(yvec == 4);  N4 = length(ind4)

#########################################################################
#                                                                      ##
# Plotting the basic details for the data set                          ##
#                                                                      ##
#########################################################################
 plot(x=0, y=0, ty="n", xlim=c(0,0.4), ylim = c(0,p),
      xlab="Average test error rate", ylab="Number of features used")
  #zoomed in point xlim=0.02 ylim=150

 ## Plotting horizontal line at p=N ##
   abline(h=N, lty="dashed")
   mtext(text="N",side=2,at=N)            #label p=N line

 ## Plotting the majority vote ##
```

```
    majority=(N-max(N1,N2,N3,N4))/N
    abline(v=majority, lty="dotted")
    mtext(text="Majority",side=1,at=majority)  #label majority vote line

##########################################################################
#                                                                       ##
# Plotting the KNN output from PMain_mult                               ##
#                                                                       ##
##########################################################################
  knn.feat = knn_out[,1]
  knn.error = knn_out[,2]
  knn.stderr = knn_out[,3]

  cols = c("red","chocolate","yellowgreen","green3","turquoise","blue",
           "purple")
  points(y=knn.feat, x=knn.error, pch=c(rep(24,6),17), col=cols,
         lwd=knn.stderr*3000)

##########################################################################
#                                                                       ##
## Plotting the NSC output from Pmain_mult                              ##
#                                                                       ##
##########################################################################
  nsc.feat = nsc_out[,1]
  nsc.error = nsc_out[,2]
  nsc.stderr = nsc_out[,3]
  colnsc = c("magenta")
  points(y=nsc.feat, x=nsc.error, pch=19, col=colnsc,lwd=nsc.stderr*3000)

 ## LEGEND ##
  names = rownames(knn_out)
  legend(x=0.275, y=2200, legend=c(names,"NSC"), col= c(cols,colnsc),
         pch=c(rep(17,7), 19))

}
```

B.30 Plotting the KNN results from PMain for the binary data sets

```
> fix(PFigure_knn)
```

```
function(data.xy, knn_outA, knn_outB, knn_outC,Kvec) {
    # Kvec= vector containing 3 possible values for the K in KNN
    # knn_outA= output from the PMain programme on binary data for K=1
    # knn_outB= output from the PMain programme on binary data for K=3
    # knn_outC= output from the PMain programme on binary data for K=5

##########################################################################
#                                                                       ##
# This program plots a graph based on the KNN output from PMain         ##
#                                                                       ##
##########################################################################

 ## Getting details for the plot skeleton ##
    data.x = as.matrix(data.xy[,-ncol(data.xy)])
    yvec = data.xy[,ncol(data.xy)]
    p = ncol(data.x)
    N = nrow(data.xy)
    ind0 = which(yvec == 0)
    ind1 = which(yvec == 1)
    N0 = length(ind0)
    N1 = length(ind1)
#########################################################################
#                                                                     ##
# Plotting the basic details for the data set                         ##
#                                                                     ##
#########################################################################
  ## Creating Space for the legend outside the plot ##
    par(xpd=NA,oma=c(3,0,0,0))

    plot(x=0, y=0, ty="n", xlim=c(0,p), ylim = c(0,0.5),
         ylab="Test Error", xlab="Number of Features used")

   ## Plotting vertical line at p=N ##
    abline(v=N, lty="dashed")
    mtext(text="N",side=1,at=N)            #labelling p=N line

   ## Plotting the majority vote ##
    majority=min(N1,N0)/N
    abline(h=majority, lty="dotted")
    mtext(text="Majority",side=2,at=majority)  #labelling majority vote

#########################################################################
#                                                                     ##
# Plotting the KNN output from PMain for all 3 Kvec values            ##
#                                                                     ##
#########################################################################
 cols = c("red","chocolate","yellowgreen","green3","turquoise","blue",
          "royalblue","purple","magenta")

   ## Plotting knn_outA ##
     points(x=knn_outA[,1], y=knn_outA[,2], pch=22, bg=cols)

   ## Plotting knn_outB ##
     points(x=knn_outB[,1], y=knn_outB[,2], pch=21, bg=cols)

   ## Plotting knn_outC ##
     points(x=knn_outC[,1], y=knn_outC[,2], pch=24, bg=cols)

  ## KNN Legend ##
    names = rownames(knn_outA)
    legend("topright", legend=names, text.col= cols, pch="")
    legend(par("usr")[1],par("usr")[3],legend=c("K=1","K=3","K=5"),
           pch=c(0,1,2),horiz=TRUE)
}
```

B.31 Plotting the SVM results from PMain for the binary data sets

```
> fix(PFigure_svm)
```

```
function(data.xy, svm_outA, svm_outB, svm_outC,Cpar) {
    #Cpar= vector containing 3 values for the C parameter in RBF SVM
    #svm_outA= output from the PMain programme on binary data for C=1
    #svm_outB= output from the PMain programme on binary data for C=1000
    #svm_outC= output from the PMain programme on binary data for C=10000

#########################################################################
#                                                                     ##
# This program plots a graph based on the SVM output from PMain       ##
#                                                                     ##
#########################################################################
```

```
 ## Getting details for the plot skeleton ##
   data.x = as.matrix(data.xy[,-ncol(data.xy)])
   yvec = data.xy[,ncol(data.xy)]
   p = ncol(data.x)
   N = nrow(data.xy)
   ind0 = which(yvec == 0);    N0 = length(ind0)
   ind1 = which(yvec == 1);    N1 = length(ind1)

#########################################################################
#                                                                      ##
# Plotting the basic details for the data set                          ##
#                                                                      ##
#########################################################################

  ## Creating Space for the legend outside the plot ##
   par(xpd=NA,oma=c(3,0,0,0))
   plot(x=0, y=0, ty="n", xlim=c(0,p), ylim = c(0,0.5),
        ylab="Test Error", xlab="Number of Features used")

  ## Plotting vertical line at p=N ##
  abline(v=N, lty="dashed")
  mtext(text="N",side=1,at=N)            #labelling p=N line

  ## Plotting the majority vote ##
  majority=min(N1,N0)/N
  abline(h=majority, lty="dotted")
  mtext(text="Majority",side=2,at=majority)  #labelling majority vote

#########################################################################
#                                                                      ##
# Plotting the SVM output from PMain for all 3 Cpar values             ##
#                                                                      ##
#########################################################################
  cols = c("red","chocolate","yellowgreen","green3","blue","royalblue",
           "purple","magenta")

  ## Plotting svm_outA ##
   points(x=svm_outA[,1], y=svm_outA[,2], pch=22, bg=cols)

  ## Plotting svm_outB ##
   points(x=svm_outB[,1], y=svm_outB[,2], pch=21, bg=cols)

  ## Plotting svm_outC ##
   points(x=svm_outC[,1], y=svm_outC[,2], pch=24, bg=cols)

 ## SVM Legend ##
  names = rownames(svm_outA)
  legend("topright", legend=names, text.col= cols, pch="")

  legend(par("usr")[1],par("usr")[3],legend=c("C=1","C=1000","C=10000"),
         pch=c(0,1,2),horiz=TRUE)
}
```

B.32 Plotting the frequently selected variabels and their selection percentage

```
> fix(PFigure_var.imporance)
```

```
function(data.xy,vsmat,percentage) {
 # vsmat = matrix indexing the variables selected from PMain
 # percentage = indicates the % of how many times a variable must be
                appear to be included in the plot

#########################################################################
#                                                                     ##
# This program plots the frequency of NB variables selected by        ##
# a) correlation thresholding                                         ##
# b) two-sample t-test VS procedure &                                 ##
# c) NSC VS procedure                                                 ##
#                                                                     ##
#########################################################################

## Obtaining basic information from data.xy ##
 ncol = ncol(data.xy)
 p = ncol-1
 N = nrow(data.xy)
 xmat = as.matrix(data.xy[,-ncol])
 yvec = data.xy[,ncol]
 nm = nrow(vsmat)

#########################################################################
#                                                                     ##
# Determining each variable in data.xy's selection frequency          ##
#                                                                     ##
#########################################################################

## Determining which variables are never selected ##
  irrel = which(apply(vsmat,2,sum)==0)
  no.irrel = length(irrel)

## Determining which variables are selected more than freq times ##
  freq = percentage*nm
  rel = which(apply(vsmat,2,sum)>freq)
  no.rel = length(rel)
  rel.mat = matrix(rep(0), ncol= no.rel, nrow=2)
  rel.mat[1,] = rel
  rel.mat[2,] = apply(vsmat[,rel],2,sum)

## Determining which variables are selected in every split##
  NB.var = which(apply(vsmat,2,sum)==nm)
  no.NBvar = length(NB.var)

#########################################################################
#                                                                     ##
# Plotting the relevant variables from data.xy selected in PMain      ##
#                                                                     ##
#########################################################################

 barplot(height=rel.mat[2,]/nm*100,horiz = TRUE,names.arg=rel.mat[1,],
las=1,cex.names=0.8,
         xlab="Percentage (%)",ylab="Variable Index")

  #horiz = TRUE --> plots bars horizontally
  #las =1 --> axis labels are all horizontal
  #names.arg = provides the bar names
  #cex.names= 0.8 --> make y-axis labels smaller
  # xlim=c(freq,nm) --> wont work :(


 return(list=c("p"=p, "Number of Irrelevant var" = no.irrel,
               "Irrelevant var" = irrel,
               "Number of relevant var" = no.rel,
               "Relevant var" = rel,
               "Number of variables which appear every time"=no.NBvar,
               "Variables which appear every time"=NB.var))
}
```

B.33 Comparison of PCA with SPCA

```
> fix(PFigurePCAvsSPCA)
```

```
function(data.xy,pcathresh) {
##########################################################################
#                                                                       ##
# This program compares PCA with SPCA and outputs a graph               ##
# The following procedures are applied:                                 ##
#   Ordinary principal components extracted from the data               ##
#   Supervised PCs based on correlation thresholding                    ##
#                                                                       ##
##########################################################################

##########################################################################
#                                                                       ##
# Obtaining data properties & Pre-processing  the data                  ##
# Obtain the row indices identifying the Y = 0 and Y = 1 data cases     ##
#                                                                       ##
##########################################################################
    ncol = ncol(data.xy)
    p = ncol-1
    N = nrow(data.xy)
    xmat = as.matrix(data.xy[,-ncol])
    yvec = data.xy[,ncol]
    meanx = apply(xmat, 2, mean)
    sdx = apply(xmat, 2, sd)
    data.x = scale(xmat, center = meanx, scale = sdx)
    data.y = yvec
    dataxy = cbind(data.x,data.y)
    ind0 = which(yvec == 0);   N0 = length(ind0)
    ind1 = which(yvec == 1);   N1 = length(ind1)

  # Obtain the 5 possible correlation threshold values
    corvec = abs(cor(data.y,data.x))
    corthr  = seq(from=0.1, to=0.8*max(corvec), length=5)

##########################################################################
#                                                                       ##
# Ordinary principal components extracted from the data                 ##
#                                                                       ##
##########################################################################

   pcaout=preProcess(data.frame(data.x),method=c("pca"),thresh=pcathresh)
   number.of.comp = pcaout$numComp
   pcamat = data.x%*%pcaout$rotation
   PC.y0 = pcamat[ind0,]    #principal components for group y=0
   PC.y1 = pcamat[ind1,]    #principal components for group y=1

##########################################################################
#                                                                       ##
# Supervised PCs based on correlation thresholding                      ##
#                                                                       ##
##########################################################################
```

```
  ## Min correlation value ##
    tout = Pcor(dataxy, corthr[1])
    xindcor1 = tout$Indices
    x.reducedcor1 = data.x[, xindcor1]
    spcaout = preProcess(data.frame(x.reducedcor1), method = c("pca"),
thresh = pcathresh)
    snumber.of.comp1 = spcaout$numComp
    spcamat1 = data.x[, xindcor1] %*% spcaout$rotation

    SPC1.y0 = spcamat1[ind0,]    # SPC for group y=0
    SPC1.y1 = spcamat1[ind1,]    # SPC for group y=1

  ## Median correlation value ##
    tout = Pcor(dataxy, corthr[3])
    xindcor2 = tout$Indices
    x.reducedcor2 = data.x[, xindcor2]

    spcaout = preProcess(data.frame(x.reducedcor2), method = c("pca"),
thresh = pcathresh)
    snumber.of.comp2 = spcaout$numComp
    spcamat2 = data.x[, xindcor2] %*% spcaout$rotation

    SPC2.y0 = spcamat2[ind0,]    # SPC for group y=0
    SPC2.y1 = spcamat2[ind1,]    # SPC for group y=1

  ## 0.8* max correlation value ##
    tout = Pcor(dataxy, corthr[5])
    xindcor3 = tout$Indices
    x.reducedcor3 = data.x[, xindcor3]

    spcaout = preProcess(data.frame(x.reducedcor3), method = c("pca"),
thresh = pcathresh)
    snumber.of.comp3 = spcaout$numComp
    spcamat3 = data.x[, xindcor3] %*% spcaout$rotation

    SPC3.y0 = spcamat3[ind0,]    # SPC for group y=0
    SPC3.y1 = spcamat3[ind1,]    # SPC for group y=1


#########################################################################
#                                                                      ##
# Plotting the first two PC and SPC                                    ##
#                                                                      ##
#########################################################################
   par(mfrow=c(2,2))
   x.min =min(min(pcamat[,1]), min(spcamat1[,1]),
              min(spcamat2[,1]), min(spcamat3[,1]))
   x.max =max(max(pcamat[,1]), max(spcamat1[,1]),
              max(spcamat2[,1]), max(spcamat3[,1]))
   y.min =min(min(pcamat[,2]), min(spcamat1[,2]),
              min(spcamat2[,2]), min(spcamat3[,2]))
   y.max =max(max(pcamat[,2]), max(spcamat1[,2]),
              max(spcamat2[,2]), max(spcamat3[,2]))

  plot(PC.y1[,1],PC.y1[,2],pch=19,col=c("green"),xlim=c(x.min,x.max),
       ylim=c(y.min, y.max),xlab="PC 1",ylab="PC 2",
       main=expression(paste("PCA")))
  points(PC.y0[,1],PC.y0[,2],pch=19,col=c("red"))

  plot(SPC1.y1[,1],SPC1.y1[,2],pch=19,col=c("green"),
       xlim=c(x.min,x.max), ylim=c(y.min, y.max), xlab="SPC 1",
       ylab="SPC 2", main=expression(paste("SPCA (0.1000)")))
  points(SPC1.y0[,1],SPC1.y0[,2],pch=19,col=c("red"))

  plot(SPC2.y1[,1],SPC2.y1[,2],pch=19,col=c("green"), xlab="SPC 1",
       ylab="SPC 2", xlim=c(x.min,x.max),ylim=c(y.min, y.max),
       main=expression(paste("SPCA (median[0.1, ",0.8%*%cor[max],"] )")))
  points(SPC2.y0[,1],SPC2.y0[,2],pch=19,col=c("red"))


  plot(SPC3.y1[,1],SPC3.y1[,2],pch=19,col=c("green"),xlab="SPC 1",
       ylab="SPC 2", ylim=c(-10,10),xlim=c(-10,10),
       main=expression(paste("SPCA (",0.8%*%cor[max],")")))
  points(SPC3.y0[,1],SPC3.y0[,2],pch=19,col=c("red"))


 return(list=c("Three correlation thresholds"=c(corthr[1],corthr[3],
corthr[5]),
          "Number of PC"=number.of.comp,
          "Number of SPC min cor"=snumber.of.comp1 ,
          "Number of SPC med cor"=snumber.of.comp2 ,
          "Number of SPC max cor"=snumber.of.comp3 ,
          "Number of variables in PC"=p,
          "Number of variables in SPC cor=0.1"=length(xindcor1),
          "Number of variables in SPC med cor"=length(xindcor2),
          "Number of variables in SPC max cor"=length(xindcor3)))
}
```

B.34 Plotting the non-zero NSC shrunken differences from the SRBCT data set

```
> fix(Pnsc_plot)
```

```
function() {

##########################################################################
#                                                                       ##
# This program plots the NSC non-zero shrunken difference of the        ##
#  relevant genes in the SRCT data set                                  ##
#                                                                       ##
##########################################################################

## Determining relevant genes ##
  data.xy = srbct.xy
  vsmat=main_out_srbct[[9]]
  percentage = 0.8
  ncol = ncol(data.xy)
  p = ncol-1
  N = nrow(data.xy)
  xmat = as.matrix(data.xy[,-ncol])
  yvec = data.xy[,ncol]
  nm = nrow(vsmat)
  Glabs = c("BL","EWS","NB","RMS")     # Group labels
  nc=4 #4 groups in SRBCT

## Determining which variables are selected more than freq times ##
  freq = percentage*nm
  rel = which(apply(vsmat,2,sum)>freq)
  no.rel = length(rel)
```

```
## Performing NSC & obtaining non-zero shrunken diff. of rel. genes ##
   genenames = khan$gene.labels.imagesID[rel]
   nsc.data=srbct.xy[,c(rel,ncol(srbct.xy))]
   nsc = Pnsc_mult(data.xy1=nsc.data,delta =3.3309)
   Dmat  = t(nsc$Dmat)
   Dmat_prime=t(nsc$Dmat_prime)

## PLOTTING THE GRAPH ##
 par(mar = c(1, 6, 1, 1), col = 1)    #increasing margins
 plot(rep(2, no.rel) + Dmat_prime[,1], 1:no.rel, xlim=c(0,2*nc + 2),
      type="n",ylim = c(1, no.rel + 3), xlab= "", ylab= "", axes= FALSE)

 cols = c(2,3,4,"purple")
 g = substring(genenames, 1, 20)
 axis(side=2, at=seq(rel), labels=g,las=1, tck=0,cex=0.2)
 mtext("Image Clone ID", side=2, line=4.5) #y-axis title
 box()
 abline(h = seq(no.rel), lty = 3, col = "gray")

 jj = rep(0, no.rel)
   for (j in 1:nc) {
     segments(jj + 2*j, seq(no.rel), jj + 2*j + Dmat_prime[, j],
              seq(no.rel),col =cols[j], lwd = 4)
     lines(c(2 * j, 2 * j), c(1, no.rel), col = cols[j])
     text(2 * j, no.rel + 2, label = Glabs[j], col =cols[j])
    }
```

B.35 Plotting the binary rankings of the top 10 classification procedures

```
> fix(PFigure_rank)
```

```
function(data) {

#########################################################################
#                                                                      ##
# This program plots the top 10 ranked binary class procedures         ##
#                                                                      ##
#########################################################################
## PLOTTING THE GRAPH ##

cols = c("red","chocolate","gold","yellowgreen","green3","turquoise",
         "blue","royalblue","purple","magenta")

plot(x=data[,2],y=data[,1],col=cols, pch=19,cex=2,ylim=c(0,10),
     xlim=c(5,16),xlab="Rank According to Test Error Rate",
     ylab="Rank according to No. of Features used")

## LEGEND ##
names =c("cor-KNN (K=5)", "cor-SPCA-KNN (K=5)","cor-SVM (C=1 000)",
         "cor-SVM (C=1)","cor-SVM (C=10 000)","NSC-DLDA",
         "SVM (C=1 000)","SVM (C=1)","SVM (C=10 000)","ttest-SVM (C=1)")

legend("topright", legend=names, text.col= cols,col=cols, pch=19)
}
```