CLASSIFICATION IN HIGH DIMENSIONAL DATA USING SPARSE TECHNIQUES

By

Agrippa Stulumani

Project presented in partial fulfilment of the requirements for the degree of Master of Commerce in the Faculty of Economic and Management Sciences at the University of Stellenbosch

Supervisor: Dr. M.M.C. Lamont

April 2019

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 project, 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 project or another project.

Signature	
Initials and Surname	**Date**
A. Stulumani	January 2019

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Abstract

Traditional classification techniques fail in the analysis of high-dimensional data. In response, new classification techniques and accompanying theory have recently emerged. These techniques are natural extensions of linear discriminant analysis. The aim is to solve the statistical challenges that arise with high-dimensional data by utilising the sparse coding (Johnstone and Titterington, 2009). In this project, our focus is on the following techniques: penalized LDA-L_1, penalized LDA-FL, sparse discriminant analysis, sparse mixture discriminant analysis and sparse partial least squares.

We evaluated the performance of these techniques in simulation studies and on two microarray gene expression datasets by comparing the test error rates and the number of features selected. In the simulation studies, we found that performance vary depending on the simulation set-up and on the classification technique used. The two microarray gene expression datasets are considered for practical implementation of these techniques. The results from the microarray gene expression datasets showed that these classification techniques achieve satisfactory accuracy.

Keywords: High-dimensional data, classification, sparse coding, feature selection, dimension reduction, microarray gene expression data.

Acknowledgement

I would like to thank my supervisor Dr. M.M.C Lamont, for all the encouragement and support that he has put into helping me to complete this project. I would like to dedicate the following poem by Philo Yan to him;

In the jungle, a lion's roar,
Loud rumbling from its core.
Magnificent mane of golden brown,
He is a king but wears no crown.

Predator and enemy they have two,
Human hunters, Hyenas too.
The lion and his lioness pride,
If you cross them, run, don't hide.

The roar of a Lion,
The rolling sound of thunder,
The chase of a Lion,
His prey runs asunder.

The Lion's roar is his symbol,
Of Strength, Of Leadership, Of pride,
The raw Roar sound is tribal,
His pride, so true, abide.

Table of Contents

List of Figures

List of Tables

CHAPTER 1
INTRODUCTION

1.1 Classification

Classification involves two crucial phases. The first phase make use of an inductive learning method to predict class labels based on characteristics about the features in the datasets. These learnings generated are known as classification rules. Then in the second phase we apply the classification rules obtained from the first phase to new and unseen datasets to determining the related class of each observation. To formally defined this process, consider the available set of data, viz. $D = \{(\boldsymbol{x}_i, y_i)\};\ i = 1,2, \dots, n$, where $\boldsymbol{x}_i \in \mathbb{R}^p$ are the input feature vectors and y_i is the class membership of observation i in the data. The data is typically divided into two data sets: one for training the model and the other for testing the model. The task is to develop a classifier so that each new feature vector $(\boldsymbol{x}^*)$ is assigned to a discrete class label y. Once the classifier has been trained, different metrics can be used to evaluate the model's performance on the test set by comparing the predicted values to true class labels in the test data set.

The area of classification comprises a wide range of algorithms and techniques that share a common objective, but from different perspectives. For example, depending on the number of class labels, classification methods can be either one-class, binary or multi-class. One-class classification is the simplest form, given observations of a single class, it is used to identify out-of-class (outliers) objects (Perera and Patel, 2018). Binary classification involves assigning observations into two classes, whereas multi-class classification assigns observations into one of several classes. These classification methods can further be organized according to different criteria depending on the type of learning (supervised, unsupervised and semi-supervised); the type of model (probabilistic, non-probabilistic, generative, discriminative); the type of learning process (batch, online) and recently the size of dataset (low-dimensional, high-dimensional) (Pérez-Ortiz *et al.*, 2016).

1.2 High-Dimensional Data

Traditional statistical data analysis considers data with a relatively large sample size (n) but only a few features (p). However, due to technological advances in the collection of data, we have seen enormous amounts of high-throughput data. Some examples of technologies that

produce these enormous amounts of data are microarrays in genomics, high-throughput video sequences, electro cardiology and chemometrics (Liu, 2013). Such data sets are known as high-dimensional data and are often characterized with tens of thousands of features while only having hundreds of observations, that is: $p \gg n$. Figure 1.1 below is an example of a heat map of NCI60 microarray gene expression data with 6830 gene expressions and 64 cancer cell lines (Hastie *et al.*, 2008).

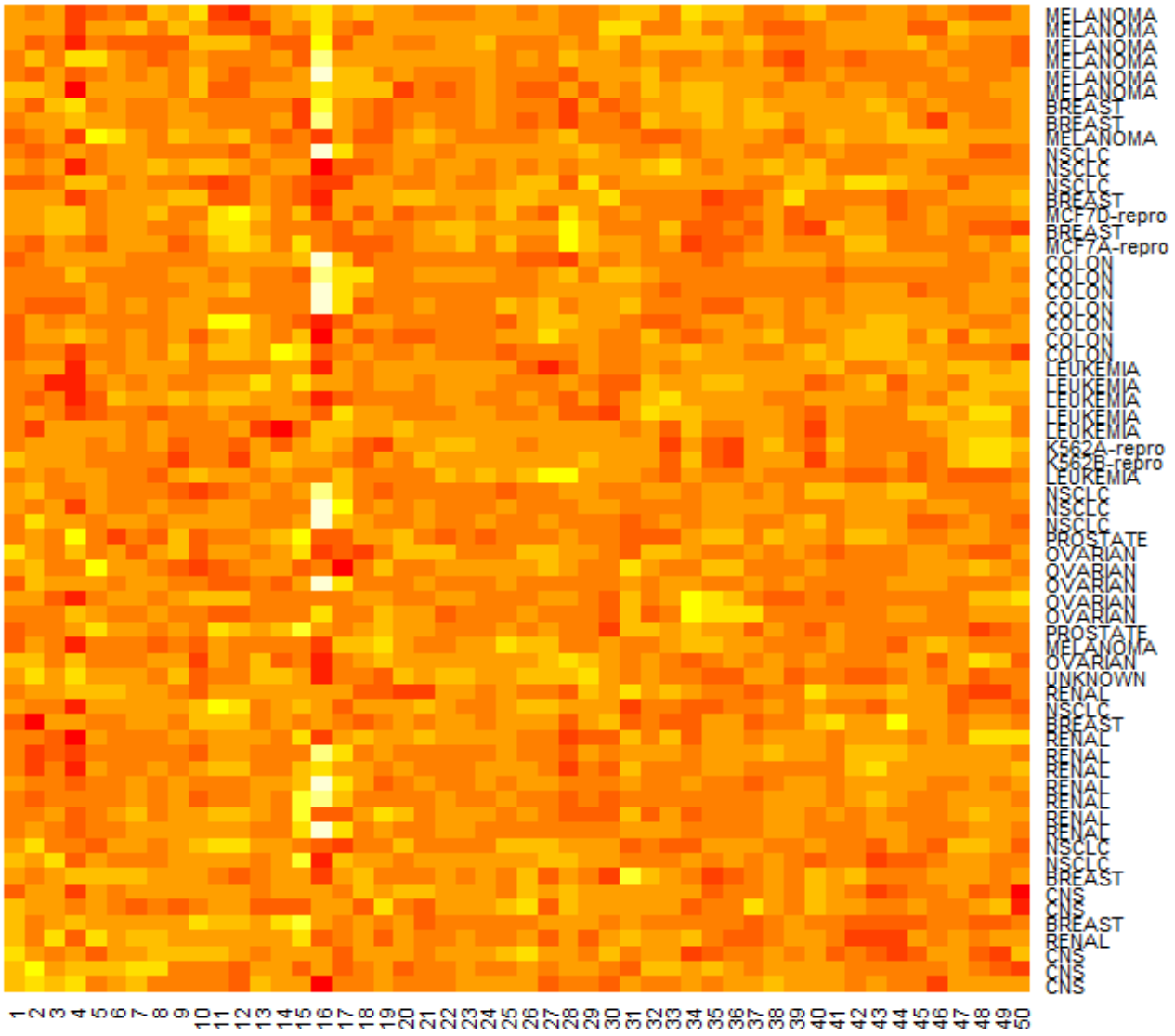

***Figure 1.1**: Heatmap generated from NCI60 microarray gene expression data. For clarity, only a random sample of 50 genes is shown. The observations are represented in rows and the genes are represented in columns. The colours varies from yellow (under expressed) to bright orange (over expressed). Missing values are shown as blanks.*

The above heat map is a graphical representation of the individual genes contained in a data matrix. From the heatmap, some genes are highly transcribed, shown by bright orange while

others are blank. This variation makes it a challenge to understand how these genes are organized. Hastie *et al.* (2008) states that these challenges means that there is a need to obtain a subset of relevant genes in order to improve both the efficiency and effectiveness of the classification algorithm which is used to classify such data. In the following section, we review feature selection as a basic approach to find a subset of features so that there is only a few features that contain the relevant information.

1.3 Feature Selection

In the high-dimensional data setting, the pervasive nature of the dimensionality of data poses serious problems to many classification algorithms with respect to scalability and learning performance (Yu and Liu, 2003). For example, there may be potentially hundreds or thousands of features. Thus, the feature space must be optimally reduced to a meaningful and manageable size. The selected subset of features should ideally possess the following characteristics, firstly the ability to cope with the curse of dimensionality; secondly, enhance generality, robustness and the level of accuracy of the model; thirdly, reducing computational costs (Destrero *et al.*, 2009).

Generally, the feature selection approach can be divided into two types: explicit and implicit feature selection. These two approaches differ on whether the (1) feature selection strategies measures feature importance independently from the learning algorithm or not and (2) whether the design of the evaluation criterion is statistical significance or cross-validation based. Below, is a brief overview of the two feature selection methods.

1.3.1 Explicit feature selection

Explicit feature selection treat the problem of finding an optimal subset of features independent of the classification algorithm. The subset selection procedure in this case can be seen as a pre-processing step to the classification algorithm (Kojadinovic and Wottka, 2000). This method assesses the degree of feature importance by looking at the intrinsic properties of the data without involving any learning algorithm. A typical explicit feature selection algorithm consists of a two-step procedure. In the first step, the algorithm evaluates and ranks features according to a certain criterion or rule. In most cases a feature importance weight/score is calculated. In the second step, low weighted or scoring features are discarded while features

with highest relevance weight or score are presented as input to the classification algorithm (Saeys *et al*., 2007).

There are various ways to measure feature importance, these could either be univariate or multivariate. In the univariate scheme, each feature is ranked independent of the full feature space while ignoring feature interactions. Examples of the univariate feature ranking criteria are: Information Gain, Pearson's Correlation Coefficient, F-Score, Chi-squared, etc. By contrast, multivariate scheme considers the dependencies between the features when performing the ranking. Examples of multivariate feature selection are entropy and mutual-information based feature selection.

The advantage of this method is that feature selection is performed only once and selected features are presented to evaluate different classifiers. Figure 1.2 below illustrates the process of feature selection for classification involving the full set of features and the final output labels are obtained from the selected features (Omar *et al.*, 2013).

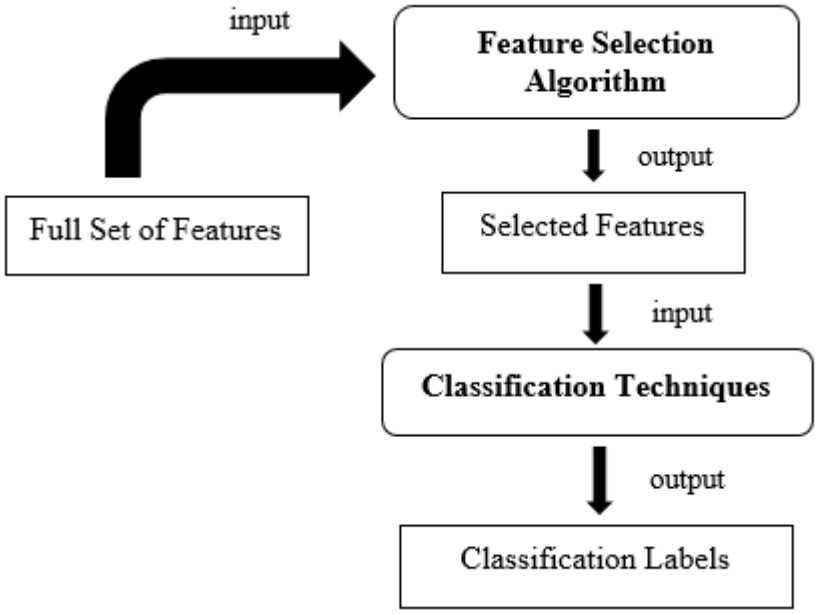

***Figure 1.2**: Explicitly feature selection schematic for classification.*

1.3.2 Implicit feature selection

The implicit methods select features directly from the learning algorithm by integrating the selection of an optimal subset of features into the classifier construction. This can be done, for instance, by optimizing the cost of the learning and the generalization ability of the model. This

implicit interaction of features with the classifier tends to generally result in better predictive power and less computational complexity than explicit methods described above, because the feature selection process is optimized for the classification algorithm to be used. For this reason, in most cases, the features selected by one implicit method might not be suitable for others (Kuang, 2009).

An example of this approach is the use of regularization penalty parameters L_1 (Lasso) and L_2 (Ridge) penalties (Efron *et al.*, 2004 and Tibshirani, 1996). The L_1 approach shrinks many of the features to zero thus encouraging sparsity, while L_2 preserves correlation (De Mol *et al.*, 2007). Any features which have non-zero coefficients are selected, making this approach a natural candidate for feature selection.

1.4 Aim and outline of this project

The aim of the project is to

- do an extensive literature review of sparse classification techniques developed for high-dimensional data such as microarray gene expression data.
- compare the classification techniques in terms of classification performance and feature selection in simulation studies and real-world microarray gene expression datasets.

The rest of the project is organised as follows: in Chapter 2, against the background of Fisher's discriminant analysis, we review the development of a series of classification techniques based on sparse coding. These techniques are natural extensions of Fisher's discriminant analysis. They are highly successful in handling multi-class classification problems in high-dimensional data. These techniques have enjoyed successful applications. For each technique reviewed, a brief description of the implementation in R is given to highlight the main features of model fitting. Chapter 3 contains simulation studies of the various techniques presented to explore the behaviour of the test error rates and the ability to perform feature selection. Chapter 4 will provide illustrative applications to show the usefulness of these model on real-world datasets. The application is demonstrated on a variety of real datasets from microarray gene expression. Chapter 5 ends with a brief discussion of the work presented in this project, as well as interpretations of important aspects and results from the project.

CHAPTER 2
SPARSE TECHNIQUES FOR CLASSIFICATION

Estimation of predictive classification models for high-dimensional data where the dimension of the feature vector is much larger than the size of the sample is becoming increasingly important. Such high-dimensional datasets present new challenges for classification methods (Xiong *et al*., 2007). As a result, new classification techniques and accompanying theory have recently emerged in response. These techniques are a natural extensions of Fisher's linear discriminant analysis. Some of these methods are: Penalized LDA-L_1, Penalized LDA - Fused Lasso, Sparse discriminant analysis, Sparse mixture discriminant analysis and Sparse partial least squares. These techniques aim to solve the statistical challenges that arise with high-dimensional data by utilising the sparsity principle (Johnstone and Titterington, 2009). Thus, effectively controlling the model complexity by simultaneous feature selection and dimension reduction.

2.1 Penalized linear discriminant analysis

2.1.1 Background

Fisher's linear discriminant analysis (LDA) is a preferred technique for supervised classification setting in many applications, due to its simplicity, robustness, and predictive accuracy. However, in the high-dimensional data setting with $p >> n$, the solution to LDA does not lead to a suitable classifier. LDA cannot be applied directly because the within-class covariance matrix of the features is singular (Witten and Tibshirani, 2011). Recently, modification to Fisher's discriminant problem have been proposed to address this problem (Clemmensen *et al.*, 2011). A more general and attractive approach was proposed by Witten and Tibshirani (2011). This approach modified Fisher's discriminant problem in two ways: (1) A diagonal estimate of the within-class class covariance matrix is used; (2) Lasso or fused lasso penalties are applied to the discriminant vectors in order to encourage sparsity and smoothness (Witten and Tibshirani, 2011).

This section is organised as follows. Section 2.1.2 gives a review of the Fisher's discriminant problem. In Section 2.1.3, we review penalized classification by using Fisher's linear

discriminant as proposed by Witten and Tibshirani (2011). Section 2.1.4 briefly discuss the implementation of Penalized LDA in R.

2.1.2 A review of Fisher's discriminant problem

Consider a classification setting. Fisher's linear discriminant analysis aims to describe n labelled observations belonging to K classes by a linear combination of features which characterizes or separates classes say C_k for $k = 1,2, \dots, K$. This is achieved either by providing the combinations of features that discriminate between classes (Fisher, 1936; Hastie *et al.*, 2008). There are several frameworks in which linear combinations can be derived. In this section, the focus is on the Fisher's discriminant framework, which seems to be the most natural approach that result in LDA (Witten and Tibshirani, 2011).

To derive the Fisher's linear discriminant classifier, consider a data matrix $\boldsymbol{X}$ that consist of n observations and p features. The features are assumed to be centred to have mean zero. Each observation $\boldsymbol{x}_i$ for $i = 1,2, \dots, n$, has a label $y_i \in \{0,1\}$, an indicator characterizing the assignment of observation $\boldsymbol{x}_i$ to class C_k. Fisher (1936) first introduced what is known as Fisher's discriminant classifier for a binary classification setting in his analysis of the iris dataset as the maximization of the ratio of the between-class covariance to the within-class covariance,

$$max_{\boldsymbol{\beta} \in \mathbb{R}^p} \left(\frac{\boldsymbol{\beta}^T \widehat{\boldsymbol{\Sigma}}_{\mathrm{b}} \boldsymbol{\beta}}{\boldsymbol{\beta}^T \widehat{\boldsymbol{\Sigma}}_{\mathrm{w}} \boldsymbol{\beta}} \right) \tag{2.1}$$

where $\boldsymbol{\beta}$ is the discriminant direction used to project the data. Furthermore, $\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}}$ and $\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}$ are the $p \times p$ maximum likelihood estimators of the within-class covariance matrix and between-class covariance matrix of the original p-dimensional data, respectively. The standard estimate of the within-class covariance matrix is defined as

$$\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}} = \frac{1}{n} \Sigma_{k=1}^{K} \Sigma_{i \in C_k} (\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_k)^T \tag{2.2}$$

where $\widehat{\boldsymbol{\mu}}_k$ is the sample mean vector of the k^{th} class and C_k is the set of observation for the k^{th} class. $\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}}$ is assumed to be non-singular. Similarly, define the between-class covariance matrix as

$$\begin{aligned}\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}} &= \frac{1}{n}\boldsymbol{X}^T\boldsymbol{X} - \widehat{\boldsymbol{\Sigma}}_{\mathrm{w}} \\ &= \frac{1}{n}\Sigma_{k=1}^{K} n_k \widehat{\boldsymbol{\mu}}_k \widehat{\boldsymbol{\mu}}_k^T \\ &= \frac{1}{n}\boldsymbol{X}^T\boldsymbol{Y}(\boldsymbol{Y}^T\boldsymbol{Y})^{-1}\boldsymbol{Y}^T\boldsymbol{X}\end{aligned} \tag{2.3}$$

where $\boldsymbol{Y}$ is an $n \times K$ matrix with y_{ik} an indicator of whether observation i is in class k (Witten and Tibshirani, 2011).

In this set-up, we are looking for a low dimensional projection such that the projected data has maximal between-class variance under a unitary constraint on the within-class variance. The Fisher's discriminant problem can thus be solved by the following objective function

$$max_{\boldsymbol{\beta}_k \in \mathbb{R}^p}\left(\boldsymbol{\beta}_k^{\mathrm{T}}\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}\boldsymbol{\beta}_k\right) \tag{2.4}$$

$$\text{subject to } \boldsymbol{\beta}_k^T\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}}\boldsymbol{\beta}_k \leq 1 \text{ and } \boldsymbol{\beta}_k^T\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}}\boldsymbol{\beta}_i = 0 \; \forall i < k$$

where $\boldsymbol{\beta}_k$ is defined as the k^{th} discriminant vector with solution $\widehat{\boldsymbol{\beta}}_k$ for $k = 1, \ldots, q < K$. When $\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}}$ has full rank, the inequality constraint in (2.4) is replaced with an equality constraint (Witten and Tibshirani, 2011). There are $K - 1$ non-trivial discriminant vectors that can be computed, these discriminant vectors summarize the original p features. From these discriminant vectors, a classification rule is obtained by computing projections $\boldsymbol{X}\widehat{\boldsymbol{\beta}}_1, \boldsymbol{X}\widehat{\boldsymbol{\beta}}_2, \ldots, \boldsymbol{X}\widehat{\boldsymbol{\beta}}_{K-1}$ and assigning each observation to its nearest centroid in the transformed space (Rao, 1948).

2.1.3 Overview of penalized LDA

As described for instance by Witten and Tibshirani (2011), in high-dimensions, the classification rule from solving (2.4) does not result to a good classifier. The classification rule for LDA involves a linear combination of all p features, it will therefore become very difficult to interpret the classifier when p becomes large, because the discriminant vectors contain p elements that have no specific structure. Also $\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}}$, the within-class covariance matrix is singular and cannot be inverted, thus, the classifier can suffer from poor performance.

Modifications of Fisher's discriminant to problem in (2.4) have been proposed. Witten and Tibshirani (2011) modified this problem by imposing a penalty function on the multiple discriminant vectors $\boldsymbol{\beta}_k$. The solution to the problem has the following general form

$$max_{\boldsymbol{\beta}_k}\left(\boldsymbol{\beta}_k^{\mathrm{T}}\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^{k}\boldsymbol{\beta}_k - P_k(\boldsymbol{\beta}_k)\right) \tag{2.5}$$

$$\text{subject to } \boldsymbol{\beta}_k^T\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}\boldsymbol{\beta}_k \leq 1$$

where $\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^{k} = \frac{1}{n}\boldsymbol{X}^T\boldsymbol{Y}(\boldsymbol{Y}^T\boldsymbol{Y})^{-1/2}\boldsymbol{P}_k^{\perp}(\boldsymbol{Y}^T\boldsymbol{Y})^{-1/2}\boldsymbol{Y}^T\boldsymbol{X}$, $\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}$ is a positive definite estimate of $\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}}$ and P_k is a penalty function on the k^{th} discriminant vector. Witten and Tibshirani (2011) defines $\boldsymbol{P}_k^{\perp} = \boldsymbol{I} - \Sigma_{i=1}^{k-1}\boldsymbol{u}_i\boldsymbol{u}_i^T$ with $\boldsymbol{u}_i$ the i^{th} singular vector of $\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}^{-1/2}\boldsymbol{X}^T\boldsymbol{Y}(\boldsymbol{Y}^T\boldsymbol{Y})^{-1/2}$. To find the k^{th} discriminant vector, problem (2.5) can be solved using the following minorization algorithm proposed by Witten and Tibshirani (2011).

Algorithm 1: Minorization algorithm to find the k^{th} discriminant vector.

(a) If $k > 1$, define an orthogonal projection matrix $\boldsymbol{P}_k^{\perp}$ that projects onto the space that is orthogonal to $(\boldsymbol{Y}^T\boldsymbol{Y})^{-1/2}\boldsymbol{Y}^T\boldsymbol{X}\widehat{\boldsymbol{\beta}}_i$ for all $i < k$. Let $\boldsymbol{P}_1^{\perp} = \boldsymbol{I}$.

(b) Let $\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^{k} = \frac{1}{n}\boldsymbol{X}^T\boldsymbol{Y}(\boldsymbol{Y}^T\boldsymbol{Y})^{-1/2}\boldsymbol{P}_k^{\perp}(\boldsymbol{Y}^T\boldsymbol{Y})^{-1/2}\boldsymbol{Y}^T\boldsymbol{X}$ with $\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^{1} = \widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}$

(c) Let $\boldsymbol{\beta}_k^{(0)}$ be the first eigenvector of $\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}^{-1}\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^{k}$, where $\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}$ is a non-singular matrix.

(d) For $m = 1, 2, \ldots$ until convergence: Let $\boldsymbol{\beta}_k^{(m)}$ be the solution to

$$max_{\boldsymbol{\beta}_k}\left(2\boldsymbol{\beta}_k^{\mathrm{T}}\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^{k}\boldsymbol{\beta}_k^{(m-1)} - P_k(\boldsymbol{\beta}_k)\right) \tag{2.6}$$

$$\text{subject to } \boldsymbol{\beta}_k^T\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}\boldsymbol{\beta}_k \leq 1.$$

Let $\widehat{\boldsymbol{\beta}}_k$ denote the solution at convergence.

The penalty term $P_k(\boldsymbol{\beta}_k)$ added to (2.6) gives a sparse classifier (Merchante, 2013). Such a sparse classifier ensures easier model interpretation and may reduce overfitting (Clemensen *et al.*, 2011). Similar to the case of Fisher's LDA, once the penalized discriminant vectors have been computed, classification is obtained by computing $\boldsymbol{X}\widehat{\boldsymbol{\beta}}_1, \boldsymbol{X}\widehat{\boldsymbol{\beta}}_2, \ldots, \boldsymbol{X}\widehat{\boldsymbol{\beta}}_q$ with $q < K - 1$ and assigning each observation to the nearest centroid in the transformed space.

Witten and Tibshirani (2011) further suggested the specific form of P_k, i.e. regular lasso (L_1 penalty) and fused lasso penalties. An L_1 penalty limits the size of each $\boldsymbol{\beta}_k$ coefficient, encouraging the solution to be sparse (i.e. a model with few coefficients) (Nowak *et al.*, 2011).

The second penalty term, a fused lasso penalty, aims to shrink the differences between every pair of class centroids for each parameter, encouraging the solution to be smooth (Guo, 2010).

2.1.3.1 Penalized LDA-L_1

In the case of an L_1 penalty, the penalized LDA-L_1 method is defined as the solution to (2.5) using

$$max_{\boldsymbol{\beta}_k}\left(2\boldsymbol{\beta}_k^T\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^k\boldsymbol{\beta}_k^{(m-1)} - \lambda_k\Sigma_{j=1}^p\left|\hat{\sigma}_j\beta_{kj}\right|\right)$$
$$\text{subject to } \boldsymbol{\beta}_k^T\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}\boldsymbol{\beta}_k \leq 1 \tag{2.7}$$

where $\hat{\sigma}_j$ is the within-class standard deviation for feature j. The size of tuning parameter λ_k is very critical, for instance, larger values corresponding to a larger penalty lead to removal of some features from the model (Shahraki *et al.*, 2015). To select an optimal tuning parameter λ_k, λ is fixed to a non-negative constant value and set

$$\lambda_k = \lambda\left\|\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}^{-1/2}\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^k\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}^{-1/2}\right\|$$

where $\|\cdot\|$ indicates the largest eigenvalue (Witten and Tibshirani, 2011). The parameters are not obtained directly, therefore, an approach for tuning λ involves cross-validating the prediction error for a grid of λ values and selecting the value that leads to the smallest cross-validation error.

2.1.3.2 Penalized LDA-FL

Witten and Tibshirani (2011) defined penalized LDA-FL by using the fused lasso penalty as a solution to (2.5) as

$$max_{\boldsymbol{\beta}_k}\left(2\boldsymbol{\beta}_k^T\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^k\beta_k^{(m-1)} - \lambda_k\Sigma_{j=1}^P\left|\hat{\sigma}_j\beta_{kj}\right| - \gamma_k\Sigma_{j=2}^P\left|\hat{\sigma}_j\beta_{kj} - \hat{\sigma}_{j-1}\beta_{k,j-1}\right|\right)$$
$$\text{subject to } \boldsymbol{\beta}_k^T\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}\boldsymbol{\beta}_k \leq 1. \tag{2.8}$$

The penalized LDA-FL requires the specification of the two penalty parameters λ_k and γ_k. The first penalty term encourages sparsity in the coefficient estimates. The second penalty term encourages sparsity in their differences, resulting in the property that the selected features occur in the adjacent groups. The parameters are not obtained directly, but by setting

$\lambda_k = \lambda\left\|\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}^{-1/2}\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^{k}\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}^{-1/2}\right\|$ and $\gamma_k = \gamma\left\|\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}^{-1/2}\widehat{\boldsymbol{\Sigma}}_{\mathrm{b}}^{k}\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}^{-1/2}\right\|$ and then finding λ and γ via cross-validation over a grid of values (Tibshirani *et al.*, 2005). Again $\|\cdot\|$ indicates the largest eigenvalue (Witten and Tibshirani, 2011). If $\gamma = 0$, the penalized LDA-L_1 procedure is obtained (Land and Friedman, 1996). A classification rule for both penalized LDA-L_1 and penalized LDA-FL is obtained similar to LDA. It is obtained by computing the projections $\boldsymbol{X}\widehat{\boldsymbol{\beta}}_1, \boldsymbol{X}\widehat{\boldsymbol{\beta}}_2, \ldots, \boldsymbol{X}\widehat{\boldsymbol{\beta}}_q$. The observations are then classified to the nearest centroid in the lower dimensional space.

2.1.4 Penalized LDA in R

The R package `PenalizedLDA` (Witten, 2015) is used to implement Penalized linear discriminant analysis. The main functions are `PenalizedLDA.cv` and `PenalizedLDA`. The function `PenalizedLDA.cv` performs cross-validation to select the optimal λ and γ used as in equation (2.7) and (2.8). The function `PenalizedLDA` solves Fisher's discriminant problem in high-dimensions using a diagonal estimate of the within-class covariance matrix and lasso or fused lasso penalties on the discriminant vectors.

Usage of the package `PenalizedLDA` in R is as follows:

```
PenalizedLDA.cv(x, y, lambda, K, nfold = 10, type =
c("standard", "ordered"), lambda2)
```

where

- `x` is an $n \times p$ data matrix.
- `y` is an n vector containing the class labels.
- `lambda` is a vector of parameters to be considered. If `type="standard"` then this is the lasso penalty tuning parameter. If `type="ordered"` then this is the tuning parameter for the sparsity component of the fused lasso penalty term.
- `K` is the number of discriminant vectors to be used.
- `nfold` is the number of cross-validation folds.
- If `type = "standard"` then a lasso penalty is used and if `type = "ordered"` the fused lasso penalty is used.

- `lambda2` is a vector of parameters to be considered. If `type="ordered"`, then this penalty controls the smoothness term in the fused lasso penalty. Larger lambda2 will lead to more smoothness.

If `cv.plda` is the output from the function above, we can get the value for λ as `cv.plda$bestlambda`, γ as `cv.plda$bestlambda2` and K as `cv.plda$bestK`.

The value for λ and γ are now known, thus the function `PenalizedLDA` can be used to perform penalized linear discriminant analysis.

To fit pernalized LDA-L_1, we use the following R command

```
PenalizedLDA(x, y, type = "standard", lambda = cv.plda$bestlambda, K = cv.plda$bestK).
```

To fit penalized LDA-FL, we use the following R command

```
PenalizedLDA(x, y, type = "ordered", lambda = cv.plda$bestlambda, K = cv.plda$bestK, lambda2 = cv.plda$bestlambda2)
```

If the output is `out.plda`, then the $\widehat{\boldsymbol{\beta}}$ coefficients are obtained by using `out.plda$discrim`.

2.2 Sparse discriminant analysis

2.2.1 Background

There are two entry points for enforcing sparsity on the Fisher's LDA framework: either directly to the LDA objective function as illustrated in (2.5) or indirectly through the optimal scoring criterion (Clemmensen *et al*., 2011; Merchante *et al*., 2012). If sparsity is enforced through optimal scoring, then the sparsity penalty is applied such that we can find a low-dimension transformation of the data that is "optimal" for classifying the cases (Hastie *et al*.,

2015). Clemmensen *et al*. (2011) developed this sparse version of LDA known as Sparse discriminant analysis.

The rest of this section is organised as follows. Section 2.2.2 gives a review of the optimal scoring criterion. In Section 2.2.3, we review the sparse optimal formulation with sparsity penalty as proposed by Clemmensen *et al.* (2011). Section 2.2.4 shows the implementation of Sparse discriminant analysis in R.

2.2.2 Optimal scoring

Suppose y_{ik} is an indicator variable where the i^{th} observation belongs to the k^{th} class,

$$y_{ik} = \begin{cases} 1 \; if \; observation \; i \; belongs \; to \; class \; k \\ 0 \; otherwise \end{cases}$$

Let $\boldsymbol{Y}$ denote the $n \times K$ indicator matrix corresponding to the dummy features for the K classes. Using this notation, the optimal scoring involves solving the sequence of problems for $k = 1, \dots, K$, each of the form

$$min_{\boldsymbol{\beta}_k \boldsymbol{\theta}_k}(\|\boldsymbol{Y}\boldsymbol{\theta}_k - \boldsymbol{X}\boldsymbol{\beta}_k\|^2) \tag{2.9}$$

$$\text{subject to } \frac{1}{n}\boldsymbol{\theta}_k^T \boldsymbol{Y}^T \boldsymbol{Y} \boldsymbol{\theta}_k = 1 \text{ and } \boldsymbol{\theta}_k^T \boldsymbol{Y}^T \boldsymbol{Y} \boldsymbol{\theta}_k = 0$$

where $\boldsymbol{X}$ is the data matrix with n observations and p features, $\boldsymbol{\theta}_k$ is a k^{th} vector of optimal scores and $\boldsymbol{\beta}_k$ is a p-dimensional discriminant vector of coefficients.

2.2.3 Overview of sparse discriminant analysis

Adding sparsity penalties to the optimal scoring criterion Clemmensen *et al.* (2011) defined Sparse discriminant analysis (SDA) by modifying (2.9) to obtain the following optimisation problem:

$$min_{\boldsymbol{\beta}_k \boldsymbol{\theta}_k}\left(\|\boldsymbol{Y}\boldsymbol{\theta}_k - \boldsymbol{X}\boldsymbol{\beta}_k\|^2 + \gamma \boldsymbol{\beta}_k^T \boldsymbol{\Omega} \boldsymbol{\beta}_k + \lambda \Sigma_{j=1}^{p} |\beta_{kj}|\right) \tag{2.10}$$

$$\text{subject to } \frac{1}{n}\boldsymbol{\theta}_k^T \boldsymbol{Y}^T \boldsymbol{Y} \boldsymbol{\theta}_k = 1 \text{ and } \boldsymbol{\theta}_k^T \boldsymbol{Y}^T \boldsymbol{Y} \boldsymbol{\theta}_k = 0 \text{ for all } j = 1,2, \dots, K-1$$

where $\boldsymbol{\Omega}$ is a positive definite matrix, $\boldsymbol{\beta}_k$ is the k^{th} discriminant vector with solution $\widehat{\boldsymbol{\beta}}_k$, $k = 1, \dots, q < K$ and $\boldsymbol{\theta}_k$ is the k^{th} vector of optimal scores. The non-negative tuning parameters γ and λ are estimated via cross-validation. From (2.10) we can view the optimal

scoring formulation as a more direct entry point for introducing sparsity into LDA models, since this formulation does not require modifications to the within-class covariance matrix (Wu *et al.*, 2015). To solve for k^{th} sparse discriminant vector $\boldsymbol{\beta}_k$ and the k^{th} scoring vector $\boldsymbol{\theta}_k$ Clemmensen *et al.* (2011) proposed an iterative algorithm where $\boldsymbol{\theta}_k$ is fixed while $\boldsymbol{\beta}_k$ is updated and then $\boldsymbol{\beta}_k$ is fixed while $\boldsymbol{\theta}_k$ is updated. This iterative process continues until convergence. The resulting discriminant vectors $\boldsymbol{\beta}_k$, will be sparse if the regularization weight λ on the L_1 penalty is sufficiently large (Hastie *et al.*, 2015). Each k^{th} discriminant vector has an independent sparse penalty, thus the sparsity profile of each discriminant vector will not be shared.

Once $\boldsymbol{\beta}_k$ has been obtained, a classification rule is obtained by first computing the projection of the vectors, $\boldsymbol{X}\widehat{\boldsymbol{\beta}}_k$. Then Fisher's LDA is performed on the $n \times q$ matrix $\boldsymbol{XB}$, with $\boldsymbol{B} = (\widehat{\boldsymbol{\beta}}_1, \ldots, \widehat{\boldsymbol{\beta}}_q)$.

2.2.4 Sparse discriminant analysis in R

The R-package `sparseLDA` (Clemmensen, 2016) provides a function for fitting Sparse discriminant analysis (SDA) classification. The function `sda()` performs sparse discriminant analysis to find the sparse discriminant vectors for linear classification. It uses an alternating minimization algorithm to minimize the SDA criterion in (2.10). Below is an example of SDA implementation in R with $\boldsymbol{X}$ matrix of the training data and $\boldsymbol{Y}$ a matrix initializing the dummy features representing the classes.

Using the package `sparseLDA`, we fit `sda()`using the following R command

```
sda(x, y, stop, maxIte = 100, Q = K-1, tol = 1e-6)
```

where

- `x` is an $n \times p$ data matrix.
- `y` is an $n \times K$ data matrix containing the dummy features representing the classes.
- `stop` is the desired number of features.
- `maxIte` is the maximum number of iterations, the default is 100 iterations.
- `Q` is number of components, the default is $K - 1$ (the number of classes minus one).

If the output is `out.sda`, the coefficient estimates of the fitted model are obtained by using `out.sda$beta`.

2.3 Sparse mixture discriminant analysis

2.3.1 Background

In most classification problems, the form of each class probability density function is unknown prior, and the selection of the algorithm that best fits the data is done over trial-and-error. Ideally, one would like to have a single formulation which can be used for most distribution types. This can be achieved by approximating the underlying distribution of each class with a mixture of normal distributions (Zhu, 2006). With this approach the strategy is to assume that there exist subclasses within each class. The j^{th} subclasses in the k^{th} class is assumed to have a multivariate normal distribution (Gkalelis *et al.*, 2011). Using this assumption, Mixture discriminant analysis is used to obtain a density estimation for each subclass.

This section is organized as follows: Section 2.3.2 gives a review of Mixture discriminant analysis (MDA). In Section 2.3.3, Sparse mixture discriminant analysis is defined as a combination of Mixture discriminant analysis and Sparse discriminant analysis as proposed by Clemmensen *et al.* (2011). Section 2.3.4 gives a brief overview of the implementation of the Sparse mixture discriminant analysis in R.

2.3.2 A review of mixture discriminant analysis

LDA tends to perform well if there are K distinct classes which are separated by linear decision boundaries. However, it can be argued that in practice linear decision boundaries are often inadequate and produces poor decision boundaries that are insufficient to separate classes. Furthermore, a single normal distribution may be insufficient in capturing the class structure (Clemmensen *et al.*, 2011). Hastie and Tibshirani (1996) proposed a mixture approach to discrimination analysis to overcome the drawbacks of LDA in this setting. The MDA is a multi-modal approach that divides each of the original classes into two or more subclasses. It then fits discriminant functions on each set of subclasses yielding a probability of membership in each subclass for each observation. The multiple distributions per class and mixture of these gives a full distribution of class. Classification is then performed using the fitted distributions (Hastie *et al.*, 2008).

To illustrate this, assume that we have n_j training observation from class k for $k = 1,2,3,\dots,K$. The subclasses are assumed to be multivariate normally distributed with the same covariance matrix, but own subclass-specific mean vector $\boldsymbol{\mu}_{jr}$. For class k, the within-class density is

$$f_k(\boldsymbol{x}) = \Sigma_{r=1}^{R_k} \pi_{kr} f(\boldsymbol{x}_i | \boldsymbol{\mu}_{kr}, \boldsymbol{\Sigma}_w) \tag{2.11}$$

where the r^{th} mixture density has prior mixing probability of π_{kr}, such that $\Sigma_{r=1}^{R_k} \pi_{kr} = 1$ where R_k is the number of subclass divisions in each class (Hastie and Tibishirani, 1996). The parameter π_{kr}, $\boldsymbol{\mu}_{kr}$ and $\boldsymbol{\Sigma}_w$ are unknown. Hastie and Tibshirani (1996) suggest employing the expectation-maximization (EM) algorithm in order to estimate these parameters. A mixture of normal distributions is estimated by varying the EM algorithm for each individual class. The within-class covariance matrix needs to be estimated by combining all the classes.

The EM algorithm alternates between two steps, the E-step and the M-step. In the E-step, given the current parameters, we compute the posterior probability that the i^{th} observation belong to the r^{th} subclass of the k^{th} class, given that it belongs to the k^{th} class:

$$p(c_{kr} | \boldsymbol{x}_i, i \in C_k) = \frac{\pi_{kr} exp(-(\boldsymbol{x}_i - \boldsymbol{\mu}_{kr})^T \boldsymbol{\Sigma}_{\mathrm{w}}^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_{kr})/2)}{\Sigma_{r'=1}^{R_k} \pi_{kr'} exp(-(\boldsymbol{x}_i - \boldsymbol{\mu}_{kr'})^T \boldsymbol{\Sigma}_{\mathrm{w}}^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_{kr'})/2)}, r = 1, \dots, R_k. \tag{2.12}$$

Then, in the M-step, the weighted maximum likelihood estimators for mixing probabilities are estimated, as well as the subclass mean and the pooled within-class covariance matrices. That is

$$\hat{\pi}_{kr} = \frac{\Sigma_{i \in C_k} \boldsymbol{x}_i p(c_{kr} | \boldsymbol{x}_i, i \in C_k)}{\Sigma_{r'=1}^{R_k} \Sigma_{i \in C_k} p(c_{kr'} | \boldsymbol{x}_i, i \in C_k)}, \tag{2.13}$$

$$\hat{\boldsymbol{\mu}}_{kr} = \frac{\Sigma_{i \in C_k} \boldsymbol{x}_i p(c_{kr} | \boldsymbol{x}_i, i \in C_k)}{\Sigma_{i \in C_k} p(c_{kr} | \boldsymbol{x}_i, i \in C_k)}, \tag{2.14}$$

$$\hat{\boldsymbol{\Sigma}}_{\mathrm{w}} = \frac{1}{n} \Sigma_{k=1}^{K} \Sigma_{i \in C_k} \Sigma_{r=1}^{R_k} p(c_{kr} | \boldsymbol{x}_i, i \in C_k)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_{kr})(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_{kr})^T. \tag{2.15}$$

The parameters for each of the normal distributions within each of the classes are estimated using the posterior probabilities from the E-step. This process continues by iterating between the E-step and the M-step until convergence, for more detail of the EM algorithm see Hastie and Tibshirani (1996).

2.3.3 Overview of sparse mixture discriminant analysis

Sparse mixture discriminant analysis (SMDA) is proposed by Clemmensen *et al.* (2011) as a combination of MDA and SDA. Clemmensen *et al.* (2011) proposed that by substituting $\boldsymbol{Y}$, an indicator matrix corresponding to the dummy features for the K class membership in SDA with a matrix of probabilities $\boldsymbol{Z}$, the SMDA can be derived. We define $\boldsymbol{Z}$, an $n \times R$ matrix of subclass probabilities as $\left(p(c_{k1}|\boldsymbol{x}_i, i \in C_k), p(c_{k2}|\boldsymbol{x}_i, i \in C_k), \dots, p\left(c_{kR_k}|\boldsymbol{x}_i, i \in C_k\right)\right)$. Using $\boldsymbol{Z}$ instead of $\boldsymbol{Y}$, we perform SDA in order to find a sequence of the k^{th} pairs of score vectors and discriminant vectors $(\boldsymbol{\theta}_k, \boldsymbol{\beta}_k)$. For the MDA, instead of using the raw data $\boldsymbol{X}$ in performing the EM iteration, we use the transformed data $\widetilde{\boldsymbol{X}} = \boldsymbol{XB}$, where the discriminant vector $\boldsymbol{B} = (\boldsymbol{\beta}_1, \dots, \boldsymbol{\beta}_q)$ for $q < R$ and is obtained from SDA. We then compute the estimated mixing probabilities, weighted means and the covariance as in equations (2.12) – (2.15) above, by using the computed projections $\widetilde{\boldsymbol{X}}$. That is

$$\tilde{\pi}_{kr} = \frac{\Sigma_{i \in C_k} \boldsymbol{x}_i p(c_{kr}|\tilde{\boldsymbol{x}}_i, i \in C_k)}{\Sigma_{r'=1}^{R_k} \Sigma_{i \in C_k} p\left(c_{kr'}|\tilde{\boldsymbol{x}}_i, i \in C_k\right)}, \tag{2.16}$$

$$\tilde{\boldsymbol{\mu}}_{kr} = \frac{\Sigma_{i \in C_k} \tilde{\boldsymbol{x}}_i p(c_{kr}|\tilde{\boldsymbol{x}}_i, i \in C_k)}{\Sigma_{i \in C_k} p(c_{kr}|\tilde{\boldsymbol{x}}_i, i \in C_k)}, \tag{2.17}$$

$$\widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}} = \frac{1}{n} \Sigma_{k=1}^{K} \Sigma_{i \in C_k} \Sigma_{r=1}^{R_k} p(c_{kr}|\boldsymbol{x}_i, i \in C_k)(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_{kr})(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_{kr})^T. \tag{2.18}$$

Now compute the subclass probabilities by using the current estimates of the mixing probabilities, weighted means and the covariance matrix and substituting $\boldsymbol{X}$ with $\widetilde{\boldsymbol{X}}$ to obtain the following

$$p(c_{kr}|\tilde{\boldsymbol{x}}_i, i \in C_k) = \frac{\pi_{kr} exp(-(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_{kr})^T \widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}^{-1} (\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_{kr})/2)}{\Sigma_{r'=1}^{R_k} \pi_{kr'} exp(-(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_{kr'})^T \widetilde{\boldsymbol{\Sigma}}_{\mathrm{w}}^{-1} (\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_{kr'})/2))}, r = 1, \dots, R_k. \tag{2.19}$$

Using the subclass probabilities in (2.12) above, update the response matrix $\boldsymbol{Z}$. Lastly, a classification rule is obtained by first computing the projections $\widetilde{\boldsymbol{X}} = \boldsymbol{XB}$ where $= \left(\widehat{\boldsymbol{\beta}}_1, \dots, \widehat{\boldsymbol{\beta}}_q\right)$ and then perform MDA on $\widetilde{\boldsymbol{X}}$.

2.3.4 Sparse mixture discriminant analysis in R

The R-package `sparseLDA` (Clemmensen, 2016) provides a function for fitting SMDA classification. The function `smda()` is used to find sparse directions for linear classification of mixture of normal models. It uses an iterative EM algorithm to find the maximum likelihood estimators and subclass probabilities as described in equations (2.12) - (2.15). Below is an example of the implementation of SMDA in R with $\boldsymbol{X}$ a data matrix of the training data and $\boldsymbol{Y}$ a matrix initializing the dummy features representing the groups. $\boldsymbol{Z}$ is a matrix initializing the probabilities representing the classes.

Using the package `sparseLDA`, we fit `smda()` using the following R command

```
smda(x, y, Z, Rj, lambda = 1e-6, maxIte = 100, Q = R-1, tol = 1e-4)
```

where

- `Rj` is a vector of the number of subclasses per class.
- `maxIte` is the maximum number of iterations, the default is 100 iterations.
- `Q` is number of components, the default is $K - 1$ (the number of classes minus one).

If the output is `out.smda`, the coefficient estimates of the fitted model are obtained by using `out.smda$beta`.

2.4 Sparse partial least squares

2.4.1 Background

Partial least squares (PLS) derives a set of latent predictors by simultaneous decomposition of the response and features. Although PLS has been shown to achieve good predictive performance, it is not particularly suitable for feature selection and therefore often produces linear combinations of the original predictors that are hard to interpret due to high dimensionality (Chung and Keleş, 2010). Chung and Keleş (2010) developed a sparse version of the PLS-based classification methods using sparse partial least squares (SPLS). Sparse partial least squares' objective is to achieve feature selection and dimension reduction simultaneously. This is achieved by imposing sparsity on the reduction step to PLS.

The rest of the section is organized as follows. Section 2.4.2 gives a review of the general principles of the partial least squares methodology. In Section 2.4.3, we review the sparse partial least squares classification methodology developed by Chun and Keleş (2010). Section 2.4.4 give a brief overview of the implementation of the sparse partial least squares classification in R.

2.4.2 Standard partial least squares

Partial least squares (PLS) is a dimension reduction technique developed by Wold (1966) for application in high-dimensional classification problems that exhibit the added complication of multicollinearity among the predictors $X_1, X_2, \dots, X_p$. At its core is a dimension reduction technique that operate under the assumption of a basic latent decomposition of the centred response matrix $\boldsymbol{Y}$ and a predictor matrix $\boldsymbol{X}$. The underlying assumption is that the observed data is generated by a system which is driven by projecting the original data onto a small number of latent features (Rosipal and Krämer, 2006). To specify the latent matrix $\boldsymbol{T} = \boldsymbol{XW}$, PLS requires finding $\boldsymbol{W} = \boldsymbol{ww^T}$, a matrix of the direction vectors, $\boldsymbol{w} = \left[w_1, w_2, \dots, w_p\right]^T$. The direction vectors capture variance in the input space, while simultaneously maximising correlation with the response $\boldsymbol{Y}$ (Frank and Friedman, 1993). As proposed by Chung and Keleş (2010) the estimates of the k^{th} direction vector $\boldsymbol{w}_k$ is obtained by solving the following optimization problem,

$$max_{\boldsymbol{w}}(\boldsymbol{w}^T\boldsymbol{M}\boldsymbol{w}) \tag{2.20}$$

$$\text{subject to } \boldsymbol{w}^T\boldsymbol{w} = 1 \text{ and } \boldsymbol{w}^T\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}}\widehat{\boldsymbol{w}}_i = 0 \; i = 1,2, \dots, K-1$$

where $\boldsymbol{M} = \boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{X}$ and $\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}}$ represents the covariance matrix of data matrix $\boldsymbol{X}$.

After the estimation of the direction vectors, classification can be performed using the resulting latent features, $\widehat{\boldsymbol{T}}_i = \widehat{\boldsymbol{w}}_i^T\boldsymbol{X}, i = 1,2,3, \dots, q < K$. The number of latent features is often smaller than the number of predictors, thus the problems of high dimensionality and the likelihood of overfitting have been addressed. Although the dimensionality of the data has been reduced from p to q, however, all predictors are assigned a non-zero weight, thus feature selection is not performed (Chun and Keleş, 2010). In order to perform feature selection, a variant of PLS called Sparse PLS (SPLS) has been proposed. SPLS utilize the sparsity principle to achieve

feature selection. Sparsity is imposed in the midst of the dimension reduction step to achieve both dimension reduction and feature selection simultaneously.

2.4.3 Overview of sparse partial least squares

While partial least squares achieve an effective reduction in the dimensionality of the data, Chun and Keleş (2010) illustrate that feature selection is not performed since each of the latent components is a linear combination of the full input set. Irrelevant features will ultimately deteriorate the performance of the partial least squares procedure. As a solution, imposing sparsity during dimension reduction step results in latent components depend only on a subset of the original set of features, thus achieving both dimension reduction and feature selection (Chun and Keleş, 2010). This provide the opportunity for parsimonious classification, improved interpretability and reduced correlation amongst the features.

Sparsity in the direction vectors resulting in latent features depending only on a subset of the original set of predictors is achieved by introducing L_1 penalty imposed on a surrogate of the direction vector, $\boldsymbol{c} = [c_1, c_2, \dots, c_p]^T$. In addition, an L_2 penalty is imposed on the surrogate vector to address the potential singularity of the matrix $\boldsymbol{M} = \boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{X}$ when solving for the surrogate of the direction vector (Chun and Keleş, 2010). An estimate of the k^{th} SPLS direction can be obtained by solving the optimisation problem:

$$max_{\boldsymbol{w},\boldsymbol{c}}\left(-\kappa \boldsymbol{w}^T\boldsymbol{M}\boldsymbol{w} + (1-\kappa)(\boldsymbol{c}-\boldsymbol{w})^T\boldsymbol{M}(\boldsymbol{c}-\boldsymbol{w}) + \lambda_1\Sigma_{j=1}^{p}|c_j| + \lambda_2\Sigma_{j=1}^{p}c_j^2\right) \qquad (2.21)$$

$$\text{subject to } \boldsymbol{w}^T\boldsymbol{M}\boldsymbol{w} = 1 \text{ and } \boldsymbol{w}^T\widehat{\boldsymbol{\Sigma}}_{\mathrm{w}}\widehat{\boldsymbol{w}}_k = 0,\ k = 1,2,\dots,K-1$$

where $\kappa < 1$. The SPLS procedure in practice requires specification of the set of tuning parameters κ, λ_1, λ_2 and K. The SPLS objective function in (2.21) requires alternatively iterating between solving for $\boldsymbol{w}$ for a fixed value of the surrogate direction vector $\boldsymbol{c}$. Chun and Keleş (2010) gives the algorithm to find the solution $\boldsymbol{c}$. Solving for $\boldsymbol{c}$ after fixing $\boldsymbol{w}$ often requires a large value of the parameter λ_2, thus, SPLS is typically fit with the choice $\lambda_2 = \infty$. Setting the λ_2 parameter to infinity yields the thresholded estimator which only depends on λ_1, which induce sparsity. Therefore, we regard the threshold λ_1 and the number of latent components K to be the two key tuning parameters.

Now, classification can be performed after construction of the latent components. Since it will typically be the case that $K < n$, feature selection is inherent in the procedure. Linear classifiers

are often utilised due to their simple interpretation. For example, if the latent components are considered as inputs in the linear discriminant analysis classification procedure, a method termed SPLS discriminant analysis (SPLSDA), developed by Chung and Keleş (2010) is obtained. A classifier obtained using SPLSDA is $\widetilde{\boldsymbol{X}} = \boldsymbol{XC}$, with $\boldsymbol{C} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_q)$.

2.4.4 Sparse partial least squares in R

The R-package `spls` (Chung, *et al*., 2016) provides functions for fitting a SPLS. The function `cv.spls()` performs v-fold cross-validation to compute the pair of sparsity hyper-parameters; λ_1 and K. The range for these parameters is specified and the cross-validation procedure searches the optimal values. The parameter λ_1 should have a value between 0 and 1. The number of hidden components, K is an integer and can range between 1 and $min\{p, (v-1)n/v\}$, where p is the number of predictors and n is the observation size. Below is an example to illustrate the implementation of SPLS in R showing a 10-fold cross-validation where λ_1 is searched for between 0.1 and 0.9. The number of hidden components is searched for between 1 and 10.

Using the package `spls`, we fit a 10-fold cross-validation to compute the pair of sparsity hyper-parameters; λ_1 and K using the following R command

```
cv.spls(x, y, K = c(1:10), eta = seq(0.1,0.9,0.1), fold=10)
```

where

- `x` is an $n \times p$ data matrix.
- `y` is an $n \times K$ data matrix containing the dummy features representing the classes.

`If cv.splsda` is the output from the function above, we obtain the optimal values for λ_1 as `cv.spls$eta.opt` and K as `cv.spls$K.opt`. Using these parameters, the function `spls` can be used to perform sparse partial least squares.

To fit `spls()` the following R command is used:

```
spls(x, y, eta=cv$eta.opt, K=cv$K.opt)
```

If the output is `out.spls`, then the coefficient estimates of SPLS fitted model are obtained by using `coef(out.spls)`.

2.5 Discussion

In this chapter, we have considered classification settings in which the dataset consists of p features on n observation, each of which belongs to K classes. Linear discriminant analysis is a favourite technique for classification. However, application of LDA when $p >> n$ is infeasible because the within-class covariance matrix tends to be singular and so the algorithm fails. To overcome this problem modifications to LDA have been proposed. The modifications are through Fisher's linear discriminant problem, optimal scoring criterion and mixture of normal distributions which is used to obtain a density estimation for each class. Using these modification procedures together with sparse penalties like lasso and fused lasso encourages the discriminant vectors to be sparse and to obtain a low dimensional projection of data. The sizes of these penalties are chosen using a v-fold cross-validation. Classification is performed in the reduced dimension using the resulting sparse discriminant vectors. The observations are assigned to the nearest class centroid in the reduced dimensional space.

CHAPTER 3
SIMULATION STUDIES

The aim of this chapter is to evaluate the performance of the sparse techniques for high-dimensional classification discussed in Chapter 2. These techniques are Penalized LDA-L_1, Penalized LDA-FL, SPLSDA and SDA. We will perform simulation studies to evaluate the classification accuracy and the ability to perform feature selection. In Section 3.1, we discuss the simulation set-ups. Section 3.2 explains how the tuning parameters were selected for each technique fitted. Section 3.3 discusses the simulation results.

3.1 Simulation Set-ups

Five simulation set-ups are considered. In every set-up, the number of the observations is 2100. These observations are equally split between the two classes. Each dataset consists of 500 features. All datasets from the different set-ups have 100 observations ($n_1 = 50$, $n_2 = 50$) belonging to the training set and 2000 observations ($n_1 = 1000$, $n_2 = 1000$) to the test set. Independent features are generated for all simulations except for Simulation 4 and Simulation 5 where features are correlated. The reason for the different simulation set-ups is to cover different situations where feature selection can be applied and to determine if correlation has influence on the behaviour of the error rate. Below are the details of the five simulation set-ups.

Simulation set-up 1: In this simulation set-up, the data is generated form a multivariate normal distribution with mean class difference for the two classes, equal covariance and uncorrelated features.

The mean vector for each class is set up as follows

Class 1:

$$\boldsymbol{\mu}_1 = \left(\underbrace{0.3, \ldots, 0.3}_{100}\ \underbrace{0, \ldots, 0}_{400}\right)$$

Class 2:

$$\boldsymbol{\mu}_2 = \left(\underbrace{0,\ldots,0}_{100}\ \underbrace{0.3,\ldots,0.3}_{100}\ \underbrace{0,\ldots,0}_{300}\right).$$

The covariance matrix for both classes is the identity matrix

$$\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \mathbf{I} = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix}$$

with dimension of $\mathbf{I}$ being $p \times p$.

The dataset per class was generated from a multivariate normal distribution as follows

Class 1: $\boldsymbol{X}_1 \sim \boldsymbol{N}(\boldsymbol{\mu}_1, \mathbf{I})$

Class 2: $\boldsymbol{X}_2 \sim \boldsymbol{N}(\boldsymbol{\mu}_2, \mathbf{I})$.

Simulation set-up 2: In this simulation set-up both the mean vectors and the covariance matrices are unequal. The covariance matrix for class 2 is inflated by a factor of five.

The mean vector for each class is set up as follows

Class 1:

$$\boldsymbol{\mu}_1 = \left(\underbrace{0.7,\ldots,0.7}_{100}\ \underbrace{0,\ldots,0}_{400}\right)$$

Class 2:

$$\boldsymbol{\mu}_2 = \left(\underbrace{0,\ldots,0}_{100}\ \underbrace{0.7,\ldots,0.7}_{100}\ \underbrace{0,\ldots,0}_{300}\right).$$

The covariance matrices for each class respectively are

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix} = \mathbf{I} \text{ and } \boldsymbol{\Sigma}_2 = \begin{bmatrix} 5 & 0 & \ldots & 0 \\ 0 & 5 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 5 \end{bmatrix} = 5 \times \mathbf{I}.$$

The per class dataset is generated from a multivariate normal distribution as follows

Class 1: $\boldsymbol{X}_1 \sim \boldsymbol{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$

Class 2: $\boldsymbol{X}_2 \sim \boldsymbol{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$.

Simulation set-up 3: In this simulation set-up, the means for the two classes are equal. The features are uncorrelated. The covariance matrices are unequal, similar to simulation set-up 2.

The mean vector for both classes were defined as

$$\boldsymbol{\mu}_1 = \boldsymbol{\mu}_2 = \boldsymbol{\mu} = \left(\underbrace{0.7, \ldots, 0.7}_{100}\ \underbrace{0, \ldots, 0}_{400}\right).$$

The respective covariance matrices for the classes are

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix} = \mathbf{I} \text{ and } \boldsymbol{\Sigma}_2 = \begin{bmatrix} 5 & 0 & \ldots & 0 \\ 0 & 5 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 5 \end{bmatrix} = 5 \times \mathbf{I}.$$

The per class dataset is generated from a multivariate normal distribution as follows

Class 1: $\boldsymbol{X}_1 \sim \boldsymbol{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_1)$

Class 2: $\boldsymbol{X}_2 \sim \boldsymbol{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_2)$.

Simulation set-up 4: In this simulation set-up, the data is generated form a multivariate normal distribution with unequal means between the two classes. The features are correlated and the covariance matrices for the two classes are equal.

The mean vector for each class is set up as follows

Class 1:

$$\boldsymbol{\mu}_1 = \left(\underbrace{0.3, \ldots, 0.3}_{100}\ \underbrace{0, \ldots, 0}_{400}\right)$$

Class 2:

$$\boldsymbol{\mu}_2 = \left(\underbrace{0, \ldots, 0}_{100}\ \underbrace{0.3, \ldots, 0.3}_{100}\ \underbrace{0, \ldots, 0}_{300}\right).$$

The common covariance matrix is

$$\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.7 & \ldots & 0.7 \\ 0.7 & 1 & \ldots & 0.7 \\ \vdots & \vdots & \ddots & \vdots \\ 0.7 & 0.7 & \ldots & 1 \end{bmatrix}.$$

The per class dataset is generated as follows

Class 1: $\boldsymbol{X}_1 \sim \boldsymbol{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$

Class 2: $\boldsymbol{X}_2 \sim \boldsymbol{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$.

Simulation set-up 5: The data generated from this simulation set-up has unequal means and correlated features between the two classes. We make use of a block diagonal covariance matrix to mimic a gene expression dataset.

The mean vector for each class is set up as follows

Class 1:

$$\boldsymbol{\mu}_1 = \left(\underbrace{0.7, \ldots, 0.7}_{100} \underbrace{0, \ldots, 0}_{400}\right)$$

Class 2:

$$\boldsymbol{\mu}_2 = \left(\underbrace{0, \ldots, 0}_{100} \underbrace{0.7, \ldots, 0.7}_{100} \underbrace{0, \ldots, 0}_{300}\right).$$

The covariance structure is a block diagonal, with five blocks each of dimension 100×100. The blocks have (j, j') element $0.6^{|j-j'|}$. This covariance structure is intended to mimic gene expression data, in which genes are positively correlated within a pathway and independent between pathways (Witten and Tibshirani, 2011). The block diagonal covariance matrix given as

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{jj} & 0 & \ldots & 0 \\ 0 & \boldsymbol{\Sigma}_{jj} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \boldsymbol{\Sigma}_{jj} \end{bmatrix} \text{ where } \boldsymbol{\Sigma}_{jj} = \left\{0.6^{|j-j'|}\right\}, \text{ for } j, j' = 1, \ldots, 100.$$

The dimension of $\boldsymbol{\Sigma}$ matrix is $p \times p$.

The per class dataset is generated from a multivariate normal distribution as follows

Class 1: $\boldsymbol{X}_1 \sim \boldsymbol{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$

Class 2: $\boldsymbol{X}_2 \sim \boldsymbol{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$.

3.2 Simulation Results

For each simulation set-up, the four classification models were fitted on the training set using a range of tuning parameters. The tuning parameters for each of the classification techniques are considered as follows:

- For penalized LDA-L_1, a grid of the tuning parameter λ is specified as (0.0001, 0.001, 0.01, 0.10, 1, 10).

- For penalized LDA-FL, the same grid of λ as in penalized LDA-L_1 is used. The parameter γ is fixed at 0.4. As suggested by Witten and Tibshirani (2011), the parameter γ is fixed to avoid performing tuning parameter selection on a two-dimensional grid, as this can become computationally very expensive.

- SDA has two tuning parameters, λ_1 and λ_2. The parameter λ_1 controls the number of features used and λ_2 is used to regularize the estimate of the $\boldsymbol{\beta}$ coefficients. These parameters were found through trial and error. The parameters used are $\lambda_1 = 0.001$ and $\lambda_2 = 0.01$.

- For SPLSDA, there are two tuning parameters. λ_1, which represents the sparsity tuning parameter and K which represents the number of latent components. The parameter λ_1 is specified as the range between 0 and 1. K is an integer value between 1 and 10.

For penalized LDA-L_1, penalized LDA-FL and SPLSDA, the optimal tuning parameters were selected using 10-fold cross-validation. The models with the optimal tuning parameters were evaluated on the test set. The simulations were repeated 100 times, each repetition with a random choice of training set and test set. The simulations were executed using the functions `Simulation1()`, `Simulation2()`, `Simulation3()`, `Simulation4()` and `Simulation5()`. These functions are written in R. At each repetition, the functions report the test error rates and number of selected features (the number of non-zero coefficients associated with features). The functions can be found in Appendix A. The results are summarized in Table 3.1 below as the means and standard deviations of the test set errors and the number of selected features. The standard deviations reflect the variability or stability of the classification technique.

Table 3.1: Simulation results for simulated data showing the means and standard deviations of the test error rate and the number of selected features computed over 100 repetitions.

		Results for the following techniques:			
Simulation		Penalized LDA-L_1	Penalized LDA-FL	SPLSDA	SDA
1	Error Rates	0.0755	**0.0189**	0.0898	0.0991
	(Std. Dev)	(0.0084)	(0.0028)	(0.0125)	(0.0130)
	Number of Features	482	**310**	423	424
	(Std. Dev)	(16.2709)	(109.3953)	(114.9873)	(8.4892)
2	Error Rates	0.0779	**0.0076**	0.0798	0.0933
	(Std. Dev)	(0.0121)	(0.0023)	(0.0130)	(0.0138)
	Number of Features	480	**295**	382	410
	(Std. Dev)	(16.2819)	(90.3686)	(121.7201)	(8.9536)
3	Error Rates	0.4388	0.4926	0.4364	**0.4280**
	(Std. Dev)	(0.0086)	(0.0146)	(0.0204)	(0.0084)
	Number of Features	479	165	**38**	439
	(Std. Dev)	(15.44)	(236.29)	(96.80)	(7.33)
4	Error Rate	0.3703	0.3346	0.0073	**0.0016**
	(Std. Dev)	(0.1390)	(0.1560)	(0.0190)	(0.0010)
	Number of Features	481	404	**396**	432
	(Std. Dev)	(13.0481)	(152.9434)	(78.4276)	(7.6818)
5	Error Rates	0.0090	**0.0066**	0.0104	0.0281
	(Std. Dev)	(0.0021)	(0.0018)	(0.0048)	(0.0058)
	Number of Features	475	355	**212**	404
	(Std. Dev)	(6.9446)	(94.5406)	(67.6121)	(8.6565)

3.3 Discussion of Results

The five simulation set-ups are given in the rows of Table 3.1. The four techniques are displayed in the columns. The results reported are the mean test error rates for each simulation and technique. The standard deviation are reported in brackets. The mean and the standard deviation for the number of features selected are also reported. The smallest error rate and the smallest number of features selected are highlighted in bold. The following are some general insights gained from the simulation results.

- From Table 3.1, performance vary depending on the simulation set-up and on the classification technique used.

- For simulation 1 and simulation 2, penalized LDA-FL performed best with the smallest mean error rate and the least number of features.

- For simulation 3 and simulation 4, SDA performed well in terms of the mean error rates. SPLSDA did not perform well in terms of the mean error rate but it's sparsity constraints penalized the coefficient better than the other techniques, i.e. the number of features selected by SPLSDA is smaller, thus the technique does well at selecting the least number of features.

- For simulation 5, penalized LDA-FL has the smallest mean error rate and SPLSDA has the least number of features.

- Penalized LDA-FL, does well with mean difference, for example the technique performed well in simulation 1, simulation 2 and simulation 5. However, it did very poorly in simulation 3 and simulation 4 with equal means.

- SDA has a small edge for simulation 3 where the means for the two classes are equal.

- Both SPLSDA and SDA did well in simulation 4 where the features are correlated.

- SDA does not perform well with simulated gene expression data in simulation 5. The other classification techniques performed well with penalized LDA-FL achieving the smallest mean error rate.

To aid comparison of the classification performances of the techniques used, box plots of the test errors rates over the 100 repetitions are given in Figure 3.1. The five graphs represents the five simulation set-ups. The test error rates are reported on the vertical axis and the four techniques (as in Table 3.1) are given on the horizontal axis. From these box plots, we can see that performance vary depending on the simulation set-up and on the classification technique used.

3.4 Conclusion

The simulation study in this chapter explored the performance of the four techniques: Penalized LDA-L_1, Penalized LDA-FL, SPLSDA and SDA. Our first interest was to see their classification performance in terms of the mean error rates under different simulation set-ups. We have found that Penalized LDA-FL perform well when the class means are different, while both SPLSDA and SDA tend to have a much smaller mean error rate when features are correlated. SDA does not perform well with simulated gene expression data. The number of features selected was our second interest. These refer to the features that have non-zero coefficients when the classification technique is performed and the number of non-zero coefficients was obtain for each repetition. For these results, we saw that penalized LDA-FL and SPLSDA generally uses less number of features compared to the rest. This means that these technique shrinks more coefficients to zero and are more sparse than the others. However, as we have seen, more sparsity does not imply smaller error rates.

In the next chapter, we will apply the four techniques mentioned above to two real-world datasets. We also compare them to Random forests. Random forests is a popular classification technique that has a reputation that it generally performs very well (Louppe, 2014). It also provides results concerning feature importance, which will be used as an indicator of the number of features selected.

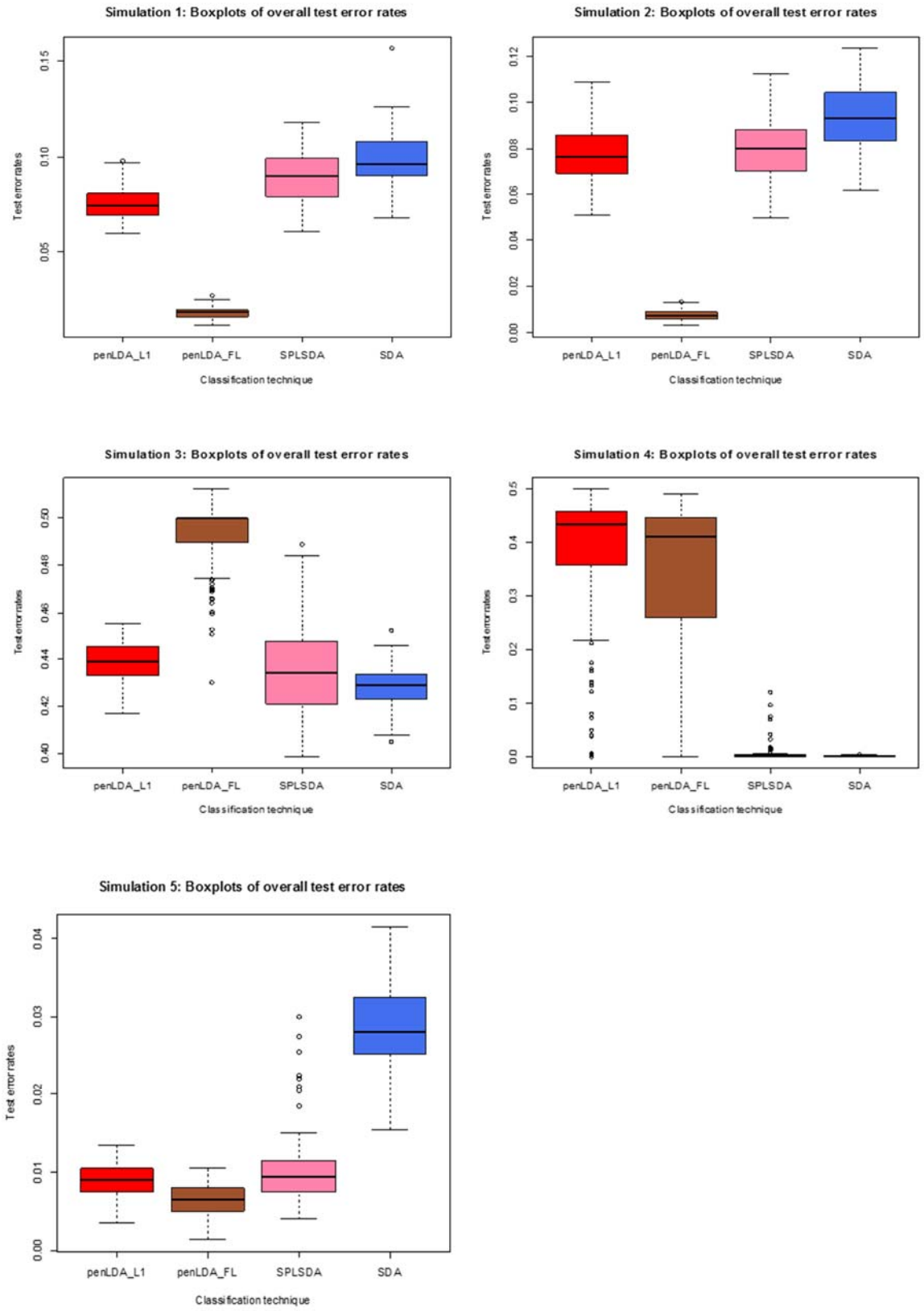

Figure 3.1: Box plots of overall test error rates for each simulation.

CHAPTER 4
EMPIRICAL STUDIES

To further evaluate the performance of the reviewed classification techniques, we consider two microarray gene expression datasets: Colon cancer dataset (Alon *et al*., 1999) and Leukemia cancer dataset (Golub *et al*., 2004). Details on the datasets are given in Section 4.2. The two datasets are high-dimensional and they involve binary classification. To determine the best performing classification technique for each dataset and to give a recommendation of the technique that can be used in future real-world applications, in addition to the test error rate, sensitivity and specificity rates are also reported. Furthermore, to evaluate and compare the effectiveness of these techniques to perform feature selection, we also fit Random forests. Random forests, proposed by Breiman (2001), is an ensemble classifier that is created by combining several classification trees and aggregating their predictions. A brief discussion of Random forests follows in the next section.

4.1 Random forests

To understand how Random forests works, we first provide some insights on how individual classification trees are constructed. Classification trees were first introduced by Breiman *et al*. (1984). Classification trees consist of a tree-structure classification rule where the root node is recursively partitioned at a series of decision nodes until the terminal node is reached. Recursive partitioning is a step-by-step process of asking an ordered sequence of questions. This process starts at the root node which consists of the entire training data set. Depending on the answers to the questions asked during the process, a tree is then constructed by either splitting or not splitting the root node. The splitting of a root node results into two daughter nodes. A daughter node can be a nonterminal or terminal node. A nonterminal is split further to form daughter nodes. A terminal node cannot split further; therefore, it is assigned a class label. The recursive partitioning process is repeated, but now the type of the questions asked at each step must depend on the answers to the previous questions of the sequence. For each node of the tree, randomly choose h features from the p input features for which to base the decision at that node. The selected features are used to determine the best split. The splitting criterion is determined by the purity of a node. To measure the purity of each node, an impurity function

is used. The impurity function used is the entropy or Gini index. The node which leads to the greatest increase in node purity is chosen (Izenman, 2008).

To generate a Random forest the algorithm uses a different training set to grow multiple trees. Each training set is obtained by taking a random sample with replacement from the original training set. Each observation in the training set has an equal probability of being selected. The remaining observations are used to estimate the test error rate of the tree, by predicting their class labels. For each of the training sets, a tree is grown by using randomly selected h features for each split. The largest tree possible is grown, that is, when all the nodes becomes pure. The grown tree is not pruned. The importance for each feature in the forest is calculated by summing up the weighted impurity decrease for all nodes where feature h is used. The sum is then divided by the number of trees in the forest to give an average. This measure is known as the Gini Importance or Mean Decrease in Impurity (Louppe, 2013). A classification is obtained by propagating an observation down each of the trees. Each tree gives a classification for the observation. Each tree of the forest provides a vote which is recorded as a classification for the observation. The votes of all trees are combined and the votes are counted. The most popular class is declared as classification of the observation.

4.2 Random forests in R

The R package `randomForest` is used to implement random forests in R. The main function is `randomForest()`. This function is used to fit a Random forest model.

Usage of the package `randomForest` in R is as follows:

```
randomForest(x, y, ntree, mtry, Importance)
```

where

- `x` is an $n \times p$ data matrix.
- `y` is an n vector containing the class labels.
- `ntree` is the number of trees to grow. The default is 500 trees.
- `mtry` is fixed number of features randomly selected at each node. The default is $\sqrt{p}$.

- Importance determine if the importance of features is to be measured.

If the output is `out.RF`, then feature importance can be obtained as `out.RF$importance`.

4.3 Microarray gene expression datasets

4.3.1 Colon cancer dataset

Colon cancer dataset was first introduced by Alon *et al*. (1999). The dataset initially had 6500 gene expression of biopsied colon epithelial cells measured on 62 patients. However, 2000 genes with the highest minimal intensity across the samples were used (Guo *et al*., 2004). Tissue samples were taken and analysed using a Affymetrix Oligonucleotide Hum6000 array. If a colon adenocarcinoma specimen was present in the tissue samples, then the sample was reported as tumorous tissue, otherwise tissue sample was reported as a normal tissue. From this analysis 40 patients were reported as tumorous and 22 patients as normal. The Colon cancer data was obtained from the `rda` package in R.

4.3.2 Leukemia cancer dataset

Leukemia cancer dataset was first introduced by Golub *et al.* (1999). This dataset contains gene expression levels from mRNA samples obtained from bone marrow or peripheral blood of 72 patients with Leukemia. The mRNA samples were split into two classes: acute lymphoblastic Leukemia (ALL) with 47 samples and acute myeloid Leukemia (AML) with 25 samples. The gene expression levels for each sample were measured using Affymetrix high-density oligonucleotide arrays containing probes for 6817 human genes (Dudoit *et al.*, 2002). The arrays had 7129 locations but only 6817 contained probes relative to human genes and the rest are controls and replicates. The Leukemia dataset underwent a pre-processing procedure described in Dudoit *et al.* (2002) to find the genes with the most variation. Only 3571 genes remained in the reduced dataset after excluding low variance genes. The Leukemia data was obtained from the `varvbs` package in R.

Table 4.1: Summary of the Colon and Leukemia microarray cancer datasets.

Dataset	**Number of features**	**Number of observations**	**Classes**
Colon	2000	62	Normal/Tumor
Leukemia	3571	72	ALL/AML

4.4 Results of empirical study

Data preparations for implementing the different classification techniques were done in the following way. Firstly, datasets were standardized so that each feature has a mean of zero and standard deviation of one. Then, the datasets were randomly split into a training set containing 70% of the observations and a test set containing the remaining 30%. During the random splitting, the proportions of the classes were kept the same as in the original data set. This is to ensure that the relationship between the different class sizes in the resulting training and test data sets was preserved. The optimal tuning parameters were found using a 5-fold cross-validation on the training set.

The tuning parameters for each of the classification technique are considered as follows:

- For penalized LDA-L_1, a grid of the tuning parameter λ is specified as (0.001, 0.01, 0.1, 1).
- For penalized LDA-FL, same grid of λ as in penalized LDA-L_1 is used. The parameter γ is fixed at 0.4.
- For SPLSDA, the parameter used are $\lambda_1 = 0.8$ and $K = 4$.
- For Random forests, the number of trees grown is 500.

The study was executed using the function `study_HD()`written in R. The procedure was repeated 50 times. At each repetition, the fitted models are evaluated on the test set. The test error rates and number of selected features were reported. The function can be found in Appendix B. The results are summarized in Table 4.2. The two datasets are given in the rows and the five techniques are given in the columns of Table 4.2. The means and standard deviations (in brackets) over the 50 repetitions are given of the test error rate and the number of features selected. To help with interpretation the smallest error rate and least number of features selected for each dataset are highlighted in bold. A discussion of the results for each dataset follows in the next two sections.

Table 4.2: Results of Colon and Leukemia cancer data showing the means and standard deviations of the test set errors and the number of features selected over 50 repetitions.

		Results for the following techniques:				
Dataset		Penalised LDA-L_1	Penalised LDA-FL	SPLSDA	SDA	Random Forest
Colon	Error Rates	**0.0282**	0.0326	0.0292	0.0288	0.0302
	(Std. Dev)	(0.0142)	(0.0137)	(0.0103)	(0.0133)	(0.0106)
	Number of Features	1807	1769	457	920	**451**
	(Std. Dev)	(12.87)	(16.27)	(51.33)	(43.58)	(41.97)
Leukemia	Error Rates	0.0072	0.0062	0.0080	0.0680	**0.0048**
	(Std. Dev)	(0.0076)	(0.0070)	(0.0061)	(0.0070)	(0.0068)
	Number of Features	3096	2958	744	1471	**458**
	(Std. Dev)	(27.81)	(43.60)	(81.75)	(44.46)	(28.29)

4.4.1 Discussion of Colon dataset results

The following is a general discussion of the results from the Colon dataset.

- The mean standard errors rates vary between 0.0103 to 0.0142, which is relatively small and indicates that there is low variability in the performance of the different classification techniques.

- The best performing classification technique is penalized LDA-L_1 with the lowest error rate of 0.0282, followed by SDA which has a mean test error rate of 0.0288 and SPLSDA with mean error of 0.0292.

- There is substantial variability in the number of features selected by SPLSDA, SDA and Random forests, standard deviation of 51.33, 43.58 and 41.97, respectively.

- Random forests has the least number of features selected with only 451 features.

- There is no correlation between number of features selected and the test error rate. For example, Random forests selected the least number of features yet performed inferior to penalized LDA-L_1. Penalized LDA-L_1 has the most selected features.

- The sparsity penalty on penalized LDA-L_1 and penalized LDA-FL penalize features in the model very lightly, this is consistent with the simulation studies.

To aid comparison of the achieved test errors rates, box plots of the error rates for the Colon dataset over 50 repetitions are given in Figure 4.1. From this graph we generally see quite a similar performance of the five techniques.

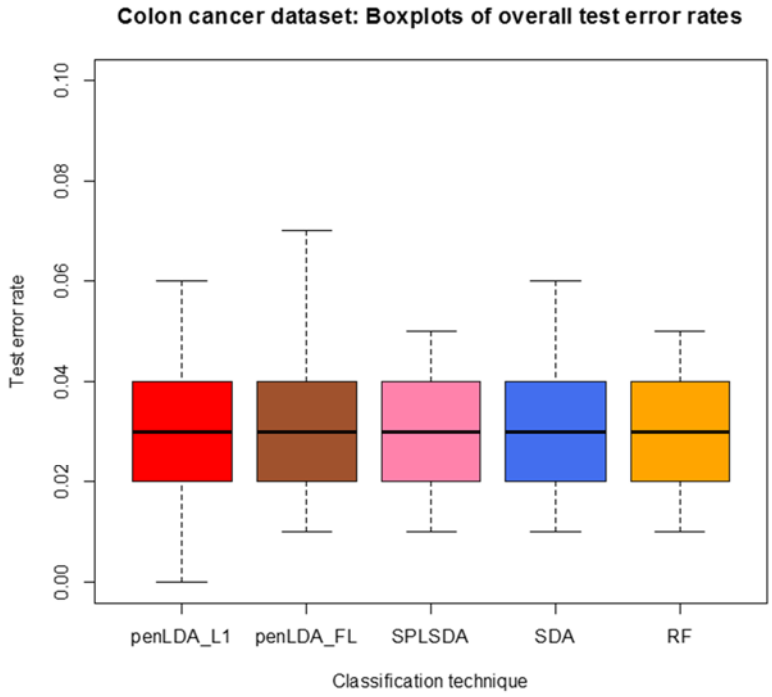

Figure 4.1: Box plots of overall test error rates for Colon cancer dataset.

4.4.2 Discussion of Leukemia dataset results

The following is a general interpretation of the results from the Leukemia dataset.

- The results in Table 4.2 shows that the Leukemia data set can be considered a low error rate dataset. The mean test error rate ranges between 0.0062 and 0.0680, which is relatively small compared to results from the Colon dataset.

- There is consistence in the degree at which the fitted techniques are penalizing the features from simulation 5 in the simulation studies, results from the Colon dataset and Leukemia dataset. For example, penalized LDA-L_1 has the most features followed by penalized LDA-*FL*, SDA, SPLSDA and Random forests.

- There is high variability in the standard deviations for the selected features, this is consistent with results from the Colon dataset and the simulation studies.

- Random forests is the best performing technique, both the mean test error rate and features selected are the smallest compared to the other techniques.

- SPLSDA has relatively the least amount of features selected, yet performed inferior to penalized LDA-L_1 and penalized LDA-FL which has the largest number of features selected.

- Again, as in the case of the Colon dataset, the sparsity penalty on penalized LDA-L_1 and penalized LDA-FL penalize features in the model very lightly. This is consistent with results from the Colon dataset and simulation studies.

To aid comparison of the test errors rates, box plots of Leukemia dataset results over 50 repetitions are given in Figure 4.2 below. From this graph we see that SDA performs very bad compared to the rest of the techniques which in turn performs quite similar.

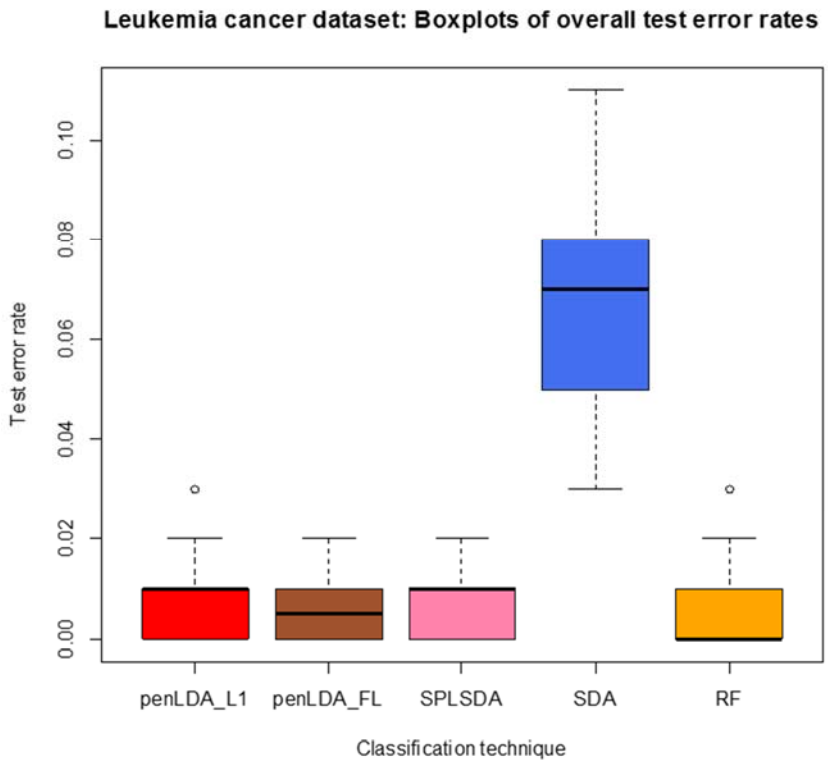

Figure 4.2: Box plots of overall test error rates for Leukemia cancer dataset.

4.4.3 Sensitivity and Specificity

Sensitivity and specificity are commonly used statistics to measure how good and reliable a diagnostic test is (Zhu *et al.*, 2010). Sensitivity evaluates how good a diagnostic test is at

detecting a disease, thus estimates the proportion of patients with a disease that are correctly identified by a diagnostic test. For instance, in the case of Colon cancer dataset, sensitivity is the probability that the biopsied colon tissue cells are classified as tumorous when the subject in fact have colon cancer. On the other hand, specificity refers to the ability of the test to correctly identify those tissues without colon cancer as such. The results of the sensitivity and specificity were also obtained for the datasets and each classification technique. The results are given in Table 4.3 and the following is a general discussion of the results.

For the Colon dataset, the techniques gave relatively high sensitivity and specificity. Penalized LDA-L_1 resulted in a much higher sensitivity. However, the specificity of the penalized LDA-L_1 is somewhat low. Random forests is doing great on the achieved sensitivity. Overall, penalized LDA-L_1 had the lowest test error rate and sensitivity compared to penalized LDA-*FL*, SDA, SPLSDA and Random forests, thus making the technique the preferred choice for classifying the colon tissue cells.

For the Leukemia dataset, the sensitivity and specificity are much more impressive than the results from the Colon dataset. All the fitted models performed well, except for SDA that has rather a poor performance. Penalized LDA-*FL* has both the highest sensitivity and specificity rate of 1.00 and 0.97, respectively. Even though Random forests gained the best test error rate, the technique did not give an overall superior performance in terms of sensitivity and specificity. The results of Random forests for sensitivity and specificity are 1.00 and 0.93 respectively which are slightly worse than penalized LDA-*FL*. However, one can still consider the Random forests as the better technique to classify the Leukemia dataset given that it has achieved the lowest error rate.

Table 4.3: Results of Colon and Leukemia cancer data showing the means and standard deviations of sensitivity and specificity computed over 50 repetitions.

		Results for the following techniques:				
Dataset		Penalized LDA-L_1	Penalized LDA-*FL*	SPLSDA	SDA	Random Forest
Colon	Sensitivity	**0.85**	0.82	0.79	0.75	0.73
	(Std. Dev)	(0.14)	(0.16)	(0.14)	(0.14)	(0.20)
	Specificity	0.82	0.85	0.88	0.85	**0.92**
	(Std. Dev)	(0.09)	(0.09)	(0.07)	(0.10)	(0.07)
Leukemia	Sensitivity	0.97	**1.00**	0.98	0.79	**1.00**
	(Std. Dev)	(0.05)	(0.01)	(0.03)	(0.11)	(0.01)
	Specificity	0.94	**0.97**	0.94	0.52	0.93
	(Std. Dev)	(0.08)	(0.06)	(0.07)	(0.20)	(0.11)

CHAPTER 5
SUMMARY

The first chapter served as an introductory chapter and provided a brief review of high-dimensional data setting and feature selection. High-dimensional dataset has enormous amounts of high-throughput data resulting to a relatively small observation size but very large number of features. Then, Chapter 2 reviewed the theoretical background for the various classification techniques used to handle high-dimensional data. These techniques adapt sparse coding to overcome the limitation of linear discriminant analysis. There are two entry points for adapting sparsity: either directly through the linear discriminant analysis objective function or through the optimal scoring criterion. Sparse coding aims to express the full set of features as a linear combination of only a subset of features. Using this subset of features, classification is performed in the reduced dimension by assigning observations to the nearest class centroid.

In Chapter 3 and 4, we have proved the statistical benefits of extending linear discriminant analysis to the high-dimensional setting by investigating the performance of different classification techniques on both simulated and real-world gene expression datasets. Simulation studies were designed to cover different situations where feature selection can be applied. Feature selection is important because the resulting discriminant vectors involve only a subset of the features. The results on the simulation studies for all five simulation studies showed that penalized LDA-FL and SPLSDA gained the sparsest solution. The results from both the Colon and Leukemia cancer datasets showed that these classification techniques achieve satisfactory accuracy.

Several problems and practical limitations remains in this project. For example, due to computational resources required and the practical complexity in choosing the parameters, there is a need for alternative approaches for selecting parameters, for example, penalized LDA-FL was performed using the parameter γ fixed at 0.4. The effect of other values of γ on classification performance could be investigated. The parameters for SDA were found through trial and error. This means that we have to go through a tedious process of many trial and error to get the best combination of parameters. There were different patterns in the test error rates for the techniques when we compare the results from the simulation study and the empirical study. The structure of the data may play an important role in these differences.

Future research:

- The simulation performed was very limited. Exploring more simulation set-ups should definitely be investigated further.
- We only studied the two-class case. Studying the multiclass case should also be investigated.
- We compared the sparse techniques to Random forests, but one may also want to compare them other techniques that does feature importance.

REFERENCES

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. *Proceedings of the National Academy of Sciences*, *96*(12), pp.6745-6750.

Breiman, L., 2001. Random Forests. *Machine Learning*, 45, pp.5-32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J., 1984. Classification and Regression Trees. Chapman & Hall/CRC.

Chun, H. and Keleş, S., 2010(a). Sparse partial least squares regression for simultaneous dimension reduction and variable selection. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, *72*(1), pp. 3–25.

Chung, D. and Keleş, S., 2010(b). Sparse partial least squares classification for high dimensional data. *Statistical applications in genetics and molecular biology*, *9*(1), Article17.

Clemmensen, L., 2016. R package to perform sparse linear discriminant analysis for gaussians and mixture of gaussians models. URL http://www.imm.dtu.dk/~lhc, https://github.com/topepo/sparselda

Destrero, A., Mosci, S., De Mol, C., Verri, A. and Odone, F., 2009. Feature selection for high-dimensional data. *Computational management science*, *6*(1), pp.25-40.

Dudoit, S., Yang, Y.H., Callow, M.J. and Speed, T. P., 2002. Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Statistica sinica*, pp.111-139.

Efron, B., Hastie, T., Johnstone, I. and Tibshirani, R., 2004. Least angle regression. *The Annals of statistics*, *32*(2), pp.407-499.

Fisher, R.A., 1936. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, *7*(2), pp.179-188.

Frank, L.E. and Friedman, J.H., 1993. A statistical view of some chemometrics regression tools. *Technometrics*, *35*(2), pp.109-135.

Gkalelis, N., Mezaris, V. and Kompatsiaris, I., 2011. Mixture subclass discriminant analysis. *IEEE Signal Processing Letters*, *18*(5), pp.319-322.

Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A. and Bloomfield, C.D., 1999. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, *286*(5439), pp.531-537.

Guo, J., 2010. Simultaneous variable selection and class fusion for high-dimensional linear discriminant analysis. *Biostatistics*, *11*(4), pp.599-608.

Guo, Y., Hastie, T. and Tibshirani, R., 2004. Regularized discriminant analysis and its application in microarrays. *Biostatistics*, *1*(1), pp.1-18.

Hastie, T. and Tibshirani, R., 1996. Discriminant analysis by Gaussian mixtures. *Journal of the Royal Statistical Society. Series B: Methodological*, pp.155-176.

Hastie, T., Tibshirani, R. and Friedman, J., 2008. *Elements of Statistical Learning – Data Mining, Inference, and Prediction,* 2nd edition, Springer.

Hastie, T., Tibshirani, R. and Wainwright, M., 2015. *Statistical learning with sparsity: the lasso and generalizations*. CRC press.

Izenman, A.J., 2008. Modern multivariate statistical techniques. *Regression, classification and manifold learning,* Springer.

Johnstone, I.M. and Titterington, D.M., 2009. Statistical challenges of high-dimensional data. *The Royal Society A: Mathematical, Physical and Engineering Sciences, 367* (1906), pp.4237-4253.

Kojadinovic, I. and Wottka, T., 2000, Comparison between a filter and a wrapper approach to variable subset selection in regression problems. *Proceedings of the European Symposium on Intelligent Techniques*, pp.311-321.

Kuang, Y., 2009. *A Comparative Study on Feature Selection Methods and Their Applications in Causal Inference*. Department of Computer Science, Faculty of Science, Lund University. Master's thesis. Available at: https://pdfs.semanticscholar.org/8da1/d82c32c13b19d7b6e515c9488da0ef0cfb3a.pdf (Accessed: 21 June 2018).

Land, S. and Friedman, J., 1996. Variable fusion: a new method of adaptive signal regression. *Tehnical Report*. Department of Statistics, Stanford University, Stanford.

Liu, T., 2013. Statistical Learning for Sample-Limited High-dimensional Problems with Application to Biomedical Data. The University of Michigan. Ph.D thesis. Available at: https://deepblue.lib.umich.edu/bitstream/handle/2027.42/97895/joyliu_1.pdf?sequence%3D1 (Accessed: 26 October 2018).

Louppe, G., Wehenkel, L., Sutera, A. and Geurts, P., 2013. Understanding variable importances in forests of randomized trees. *Advances in neural information processing systems*, pp. 431-439.

Merchante, L.F.S., 2013. *Learning algorithms for sparse classification*. PhD thesis. Université de Technologie de Compiègne. Available at: https://tel.archives-ouvertes.fr/tel-00868847

Nowak, G., Hastie, T., Pollack, J.R. and Tibshirani, R., 2011. A fused lasso latent feature model for analyzing multi-sample aCGH data. *Biostatistics*, *12*(4), pp.776-791.

Omar, N., Jusoh, F., Ibrahim, R. and Othman, M.S., 2013. Review of feature selection for solving classification problems. *Journal of Information System Research and Innovation*, *3*, pp.64-70.

Perera, P. and Patel, V.M., 2018. Learning deep features for one-class classification. *arXiv preprint arXiv:1801.05365*.

Pérez-Ortiz, M., Jiménez-Fernández, S., Gutiérrez, P.A., Alexandre, E., Hervás-Martínez, C. and Salcedo-Sanz, S., 2016. A review of classification problems and algorithms in renewable energy applications. *Energies*, *9*(8), p.1-27.

Rao, C.R., 1948. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society. Series B: Methodological*, *10*(2), pp.159-203.

Rosipal, R. and Krämer, N., 2005. Overview and recent advances in partial least squares. *International Statistical and Optimization Perspectives Workshop "Subspace, Latent Structure and Feature Selection"*, pp. 34-51.

Saeys, Y., Inza, I. and Larrañaga, P., 2007. A review of feature selection techniques in bioinformatics. *Bioinformatics*, *23*(19), pp.2507-2517.

Shahraki, H.R., Salehi, A. and Zare, N., 2015. Survival prognostic factors of male breast cancer in Southern Iran: a LASSO-Cox regression approach. *Asian Pacific Journal of Cancer Prevention*, *16*(15), pp.6773-6777.

Tibshirani, R., 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, pp.267-288.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J. and Knight, K., 2005. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, *67*(1), pp.91-108.

Witten, D., 2015. Penalized Classification using Fisher's Linear Discriminant. R package version 1.1, URL https://cran.rproject.org/web/packages/penalizedLDA/penalizedLDA.pdf

Witten, D. and Tibshirani, R., 2011. Penalized classification using Fisher's linear discriminant. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *73*(5), pp.753-772.

Wold, H., 1966. Estimation of principal components and related models by iterative least squares. *Multivariate analysis*, pp.391-420.

Wu, Y., Wipf, D. and Yun, J.M., 2015, Understanding and evaluating sparse linear discriminant analysis. *Artificial Intelligence and Statistics*, pp. 1070-1078.

Xiong, T., 2007. *Classification methods for high-dimensional sparse data*. Ph.D Dissertation. Minneapolis, MN, USA.

Xiong, T., Bit, J., Rao, B., and Cherkassky, V.S., 2007. Probabilistic joint feature selection for multi-task learning. *Society for Industrial and Applied Mathematics: International Conference on Data Mining, 11*, pp. 332-342.

Yu, L. and Liu, H., 2003. Feature selection for high-dimensional data: A fast correlation-based filter solution. *Proceeding of the international conference on machine learning*, pp. 856-863.

Zhu, M., 2006. *A study of the generalized eigenvalue decomposition in discriminant analysis*. Ph.D Dissertation. Ohio State University. Available at: http://rave.ohiolink.edu/etdc/view?acc_num=osu1152133627

Zhu, W., Zeng, N. and Wang, N., 2010. Sensitivity, specificity, accuracy, associated confidence interval and ROC analysis with practical SAS implementations. NESUG proceedings: health care and life sciences, Baltimore, Maryland, 19, p.67.

APPENDIX

A. R functions for Simulation Studies.

```
##### Simulation 1: mean difference, equal covariance and uncorrelated
features

Simulation1 <- function(){

#### Simlation1 performs sparse classification in high-dimensional data
using the following techniques

# 1. PenalizedLDA L1

# 2. PenalizedLDA FL

# 3. SDA

# 4. SPLSDA

reps = 100 # Number of simulation repatitions

result <- matrix(nrow = reps, ncol = 9)

nzerobetas <- list()

nzerobetasFL <- list()

nzerobetasSDA <- list()

nzerobetasSPLSDA <- list()

colnames(result) <- c("Repetition", "Error.L1", "NonZeroFeatures.L1",
"Error.FL", "NonZeroFeatures.FL", "Error.SPLSDA", "NonZeroFeatures.SPLSDA",

"Error.SDA", "NonZeroFeatures.SDA")

#### Simulation study for classification in high-dimensional

for (k in 1:reps) {

start.time <- Sys.time()

simulation <- function()

{

library(MASS)

library(penalizedLDA)

library(sparseLDA)
```

```
library(rgl)

library(spls)

library(beepr)


p = 500  # p = number of features

## Training data

mu1 <- matrix(c(rep(0.3, 100), rep(0, 400)))

mu2 <- matrix(c(rep(0, 100), rep(0.3, 100), rep(0, 300)))

sig <- matrix(0 ,p ,p)

diag(sig) <- 1

n1=n2=50

Class1 <- mvrnorm(n1, mu1, sig)

Class2 <- mvrnorm(n2, mu2, sig) # inflate the covariance matrix by 5

train_X <- rbind(Class1, Class2)

train_Y <- matrix(c(rep(1, n1), rep(2, n2)))

Y1 <- matrix(c(rep(1, n1), rep(0, n2)))

Y2 <- matrix(c(rep(0, n1), rep(1, n2)))

Y_train <- cbind(Y1, Y2)

colnames(Y_train) <- c("Class1", "Class2")

## Test data

m1=m2=1000

Class.1 <- mvrnorm(m1, mu1, sig)

Class.2 <- mvrnorm(m2, mu2, sig)

test_X <- rbind(Class.1, Class.2)

test_Y <- matrix(c(rep(1, m1), rep(2, m2)))


Y.1 <- matrix(c(rep(1, m1), rep(0, m2)))

Y.2 <- matrix(c(rep(0, m1), rep(1, m2)))
```

```
X_test <- cbind(Y.1, Y.2)

colnames(X_test) <- c("Class1", "Class2")


################## PenalizedLDA ###################

### If type="standard" then we make use of lasso penalties.

cvL1 <- PenalizedLDA.cv(x=train_X, y=train_Y, type="standard",
lambdas=c(1e-4, 1e-3, 1e-2, 0.1, 1, 10), nfold=10)

out <- PenalizedLDA(x=train_X, y=train_Y, type="standard",
lambda=cvL1$bestlambda, K=1)


### If type="ordered" then we make use of lasso penalties.

cvFL <- PenalizedLDA.cv(x=train_X, y=train_Y, type="ordered", lambdas=c(1e-
4, 1e-3, 1e-2, 0.1, 1, 10), lambda2=0.4, nfold=10)

outFL <- PenalizedLDA(x=train_X, y=train_Y, type="ordered",
lambda=cvFL$bestlambda, K=1, lambda2=0.4)


#################### SPLSDA ######################

### Select tuning parameter K (number of latent components)

cvSPLSDA <- cv.spls(x=train_X, y=train_Y, fold=10, K=1:10,
eta=seq(0.1,0.9,0.1), kappa=0.5, plot.it=FALSE)

outSPLSDA <- splsda(x=train_X, y=train_Y, K=cvSPLSDA$K.opt,
eta=cvSPLSDA$eta.opt, kappa=0.5, scale.x=FALSE, classifier ='lda')


#################### SDA ######################

Xc <- normalize(train_X)

Xn <- Xc$Xc

outSDA <- sda(x=Xn, y=Y_train, lambda=0.01, stop=0.001, maxIte=10,
trace=FALSE)


## Selected features
```

```
discrim <- out$discrim

discrimFL <- outFL$discrim

discrimSDA <- outSDA$beta

discrimSPLSDA <- coef(outSPLSDA)


## Count of non zero coefficients

non_zero_coef <- sum(out$discrim[,1] != 0)

non_zero_coefFL <- sum(outFL$discrim[,1] != 0)

non_zero_coefSDA <- sum(outSDA$beta[,1] != 0)

non_zero_coefSPLSDA <- sum(coef(outSPLSDA)[,1] != 0)


## Predictions

pred <- predict(out, xte=test_X)

predFL <- predict(outFL, xte=test_X)

predSDA <- ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class1", 1,ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class2", 2, 0))

predSPLSDA <- predict(outSPLSDA, newx=test_X, type="fit")


## Test Error Rates

error <- sum(test_Y != pred$ypred)/nrow(test_X)

errorFL <- sum(test_Y != predFL$ypred)/nrow(test_X)

errorSDA <- sum(test_Y != predSDA)/nrow(test_X)

errorSPLSDA <- sum(test_Y != as.matrix(sapply(predSPLSDA, function(x)
as.numeric(as.character(x)))))/nrow(test_X)


output <- (list(error=error, discrim=discrim, non_zero_coef=non_zero_coef,

errorFL=errorFL, discrimFL=discrimFL, non_zero_coefFL=non_zero_coefFL,
```

```
errorSPLSDA=errorSPLSDA, discrimSPLSDA=discrimSPLSDA,
non_zero_coefSPLSDA=non_zero_coefSPLSDA, errorSDA=errorSDA,
non_zero_coefSDA=non_zero_coefSDA)   )

return(output)

}

simulation()


result[k, 1] <- k

result[k, 2] <- simulation()$error

result[k, 3] <- simulation()$non_zero_coef

result[k, 4] <- simulation()$errorFL

result[k, 5] <- simulation()$non_zero_coefFL

result[k, 6] <- simulation()$errorSPLSDA

result[k, 7] <- simulation()$non_zero_coefSPLSDA

result[k, 8] <- simulation()$errorSDA

result[k, 9] <- simulation()$non_zero_coefSDA


## matrix of coefficients

nzerobetas[[k]] <- simulation()$discrim

nzerobetasFL[[k]] <- simulation()$discrimFL

nzerobetasSPLSDA[[k]] <- simulation()$discrimSPLSDA

nzerobetasSDA[[k]] <- simulation()$discrimSDA


coefL1 <- t(matrix(unlist(nzerobetas), ncol = p, byrow = FALSE))

coefFL <- t(matrix(unlist( nzerobetasFL), ncol = p, byrow = FALSE))

coefSPLSDA <- t(matrix(unlist(nzerobetasSPLSDA), ncol = p, byrow = FALSE))


obj <- (list(result=result, coefL1=coefL1, coefFL=coefFL,
coefSPLSDA=coefSPLSDA))
```

```
end.time <- Sys.time()

cat("Iteration", k, "out of", reps, "is completed.", "Time taken is",
(end.time - start.time), "minutes.","\n")

 beep(2)

if (k == reps) cat("Done!\n")

}

return(obj)

}

simu1 <- Simulation1()


###### Simulation 2: mean difference, unequal covariance and uncorrelated
features.

Simulation2 <- function(){

#### Simlation2 performs sparse classification in high-dimensional data
using the following techniques

# 1. PenalizedLDA L1

# 2. PenalizedLDA FL

# 3. SDA

# 4. SPLSDA

reps = 100 # Number of simulation repatitions

result <- matrix(nrow = reps, ncol = 9)

nzerobetas <- list()

nzerobetasFL <- list()

nzerobetasSDA <- list()

nzerobetasSPLSDA <- list()

colnames(result) <- c("Repetition", "Error.L1", "NonZeroFeatures.L1",
"Error.FL", "NonZeroFeatures.FL", "Error.SPLSDA", "NonZeroFeatures.SPLSDA",

"Error.SDA", "NonZeroFeatures.SDA")

#### Simulation study for classification in high-dimensional
```

```
for (k in 1:reps) {

start.time <- Sys.time()

simulation <- function()

{

library(MASS)

library(penalizedLDA)

library(sparseLDA)

library(rgl)

library(spls)

library(beepr)


p = 500  # p = number of features

## Training data

mu1 <- matrix(c(rep(0.7,100), rep(0, 400)))

mu2 <- matrix(c(rep(0,100), rep(0.7, 100), rep(0, 300)))


sig <- matrix(0 ,p ,p)

diag(sig) <- 1


n1=n2=50

Class1 <- mvrnorm(n1, mu1, sig)

Class2 <- mvrnorm(n2, mu2, 5*sig) # inflate the covariance matrix by 5

train_X <- rbind(Class1, Class2)

train_Y <- matrix(c(rep(1, n1), rep(2, n2)))


Y1 <- matrix(c(rep(1, n1), rep(0, n2)))

Y2 <- matrix(c(rep(0, n1), rep(1, n2)))

Y_train <- cbind(Y1, Y2)
```

```
colnames(Y_train) <- c("Class1", "Class2")

## Test data

m1=m2=1000

Class.1 <- mvrnorm(m1, mu1, sig)

Class.2 <- mvrnorm(m2, mu2, 5*sig) # inflate the covariance matrix by 5

test_X <- rbind(Class.1, Class.2)

test_Y <- matrix(c(rep(1, m1), rep(2, m2)))


Y.1 <- matrix(c(rep(1, m1), rep(0, m2)))

Y.2 <- matrix(c(rep(0, m1), rep(1, m2)))

X_test <- cbind(Y.1, Y.2)

colnames(X_test) <- c("Class1", "Class2")


################## PenalizedLDA ###################

### If type="standard" then we make use of lasso penalties.

cvL1 <- PenalizedLDA.cv(x=train_X, y=train_Y, type="standard",
lambdas=c(1e-4, 1e-3, 1e-2, 0.1, 1, 10), nfold=10)

out <- PenalizedLDA(x=train_X, y=train_Y, type="standard",
lambda=cvL1$bestlambda, K=1)


### If type="ordered" then we make use of lasso penalties.

cvFL <- PenalizedLDA.cv(x=train_X, y=train_Y, type="ordered", lambdas=c(1e-
4, 1e-3, 1e-2, 0.1, 1, 10), lambda2=0.4, nfold=10)

outFL <- PenalizedLDA(x=train_X, y=train_Y, type="ordered",
lambda=cvFL$bestlambda, K=1, lambda2=0.4)


################### SPLSDA ######################

### Select tuning parameter K (number of latent components)
```

```
cvSPLSDA <- cv.spls(x=train_X, y=train_Y, fold=10, K=1:10,
eta=seq(0.1,0.9,0.1), kappa=0.5, plot.it=FALSE)

outSPLSDA <- splsda(x=train_X, y=train_Y, K=cvSPLSDA$K.opt,
eta=cvSPLSDA$eta.opt, kappa=0.5, scale.x=FALSE, classifier ='lda')


#################### SDA ########################

Xc <- normalize(train_X)

Xn <- Xc$Xc

outSDA <- sda(x=Xn, y=Y_train, lambda=0.01, stop=0.001, maxIte=10,
trace=FALSE)


## Selected features

discrim <- out$discrim

discrimFL <- outFL$discrim

discrimSDA <- outSDA$beta

discrimSPLSDA <- coef(outSPLSDA)


## Count of non zero coefficients

non_zero_coef <- sum(out$discrim[,1] != 0)

non_zero_coefFL <- sum(outFL$discrim[,1] != 0)

non_zero_coefSDA <- sum(outSDA$beta[,1] != 0)

non_zero_coefSPLSDA <- sum(coef(outSPLSDA)[,1] != 0)


## Predictions

pred <- predict(out, xte=test_X)

predFL <- predict(outFL, xte=test_X)

predSDA <- ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class1", 1,ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class2", 2, 0))
```

```
predSPLSDA <- predict(outSPLSDA, newx=test_X, type="fit")


## Test Error Rates

error <- sum(test_Y != pred$ypred)/nrow(test_X)

errorFL <- sum(test_Y != predFL$ypred)/nrow(test_X)

errorSDA <- sum(test_Y != predSDA)/nrow(test_X)

errorSPLSDA <- sum(test_Y != as.matrix(sapply(predSPLSDA, function(x)
as.numeric(as.character(x)))))/nrow(test_X)


output <- (list(error=error, discrim=discrim, non_zero_coef=non_zero_coef,

errorFL=errorFL, discrimFL=discrimFL, non_zero_coefFL=non_zero_coefFL,

errorSPLSDA=errorSPLSDA, discrimSPLSDA=discrimSPLSDA,
non_zero_coefSPLSDA=non_zero_coefSPLSDA, errorSDA=errorSDA,
non_zero_coefSDA=non_zero_coefSDA)   )

return(output)

}

simulation()


result[k, 1] <- k

result[k, 2] <- simulation()$error

result[k, 3] <- simulation()$non_zero_coef

result[k, 4] <- simulation()$errorFL

result[k, 5] <- simulation()$non_zero_coefFL

result[k, 6] <- simulation()$errorSPLSDA

result[k, 7] <- simulation()$non_zero_coefSPLSDA

result[k, 8] <- simulation()$errorSDA

result[k, 9] <- simulation()$non_zero_coefSDA


## matrix of coefficients
```

```
nzerobetas[[k]] <- simulation()$discrim

nzerobetasFL[[k]] <- simulation()$discrimFL

nzerobetasSPLSDA[[k]] <- simulation()$discrimSPLSDA

nzerobetasSDA[[k]] <- simulation()$discrimSDA


coefL1 <- t(matrix(unlist(nzerobetas), ncol = p, byrow = FALSE))

coefFL <- t(matrix(unlist( nzerobetasFL), ncol = p, byrow = FALSE))

coefSPLSDA <- t(matrix(unlist(nzerobetasSPLSDA), ncol = p, byrow = FALSE))


obj <- (list(result=result, coefL1=coefL1, coefFL=coefFL,
coefSPLSDA=coefSPLSDA))

end.time <- Sys.time()

cat("Iteration", k, "out of", reps, "is completed.", "Time taken is",
(end.time - start.time), "minutes.","\n")

 beep(2)

if (k == reps) cat("Done!\n")

}

return(obj)

}

simu2 <- Simulation2()


###### Simulation 3: equal means, unequal covariance and uncorrelated
features.

Simulation3 <- function(){

#### Simlation3 performs sparse classification in high-dimensional data
using the following techniques

# 1. PenalizedLDA L1

# 2. PenalizedLDA FL

# 3. SDA
```

```
# 4. SPLSDA

reps = 100 # Number of simulation repatitions

result <- matrix(nrow = reps, ncol = 9)

nzerobetas <- list()

nzerobetasFL <- list()

nzerobetasSDA <- list()

nzerobetasSPLSDA <- list()

colnames(result) <- c("Repetition", "Error.L1", "NonZeroFeatures.L1",
"Error.FL", "NonZeroFeatures.FL", "Error.SPLSDA", "NonZeroFeatures.SPLSDA",

"Error.SDA", "NonZeroFeatures.SDA")

#### Simulation study for classification in high-dimensional

for (k in 1:reps) {

start.time <- Sys.time()

simulation <- function()

{

library(MASS)

library(penalizedLDA)

library(sparseLDA)

library(rgl)

library(spls)

library(beepr)


p = 500  # p = number of features

#Simulation 3: equal means, unequal covariance and uncorrelated features.

## Training data

mu1 <- matrix(c(rep(0.7,100), rep(0, 400)))


sig <- matrix(0 ,p ,p)
```

```
diag(sig) <- 1


n1=n2=50

Class1 <- mvrnorm(n1, mu1, sig)

Class2 <- mvrnorm(n2, mu1, 5*sig) # inflate the covariance matrix by 5

train_X <- rbind(Class1, Class2)

train_Y <- matrix(c(rep(1, n1), rep(2, n2)))


Y1 <- matrix(c(rep(1, n1), rep(0, n2)))

Y2 <- matrix(c(rep(0, n1), rep(1, n2)))

Y_train <- cbind(Y1, Y2)

colnames(Y_train) <- c("Class1", "Class2")


## Test data

m1=m2=1000

Class.1 <- mvrnorm(m1, mu1, sig)

Class.2 <- mvrnorm(m2, mu1, 5*sig)

test_X <- rbind(Class.1, Class.2)

test_Y <- matrix(c(rep(1, m1), rep(2, m2)))


Y.1 <- matrix(c(rep(1, m1), rep(0, m2)))

Y.2 <- matrix(c(rep(0, m1), rep(1, m2)))

X_test <- cbind(Y.1, Y.2)

colnames(X_test) <- c("Class1", "Class2")


################## PenalizedLDA ###################

### If type="standard" then we make use of lasso penalties.
```

```
cvL1 <- PenalizedLDA.cv(x=train_X, y=train_Y, type="standard",
lambdas=c(1e-4, 1e-3, 1e-2, 0.1, 1, 10), nfold=10)

out <- PenalizedLDA(x=train_X, y=train_Y, type="standard",
lambda=cvL1$bestlambda, K=1)


### If type="ordered" then we make use of lasso penalties.

cvFL <- PenalizedLDA.cv(x=train_X, y=train_Y, type="ordered", lambdas=c(1e-
4, 1e-3, 1e-2, 0.1, 1, 10), lambda2=0.4, nfold=10)

outFL <- PenalizedLDA(x=train_X, y=train_Y, type="ordered",
lambda=cvFL$bestlambda, K=1, lambda2=0.4)


#################### SPLSDA ######################

### Select tuning parameter K (number of latent components)

cvSPLSDA <- cv.spls(x=train_X, y=train_Y, fold=10, K=1:10,
eta=seq(0.1,0.9,0.1), kappa=0.5, plot.it=FALSE)

outSPLSDA <- splsda(x=train_X, y=train_Y, K=cvSPLSDA$K.opt,
eta=cvSPLSDA$eta.opt, kappa=0.5, scale.x=FALSE, classifier ='lda')


#################### SDA ######################

Xc <- normalize(train_X)

Xn <- Xc$Xc

outSDA <- sda(x=Xn, y=Y_train, lambda=0.01, stop=0.001, maxIte=10,
trace=FALSE)


## Selected features

discrim <- out$discrim

discrimFL <- outFL$discrim

discrimSDA <- outSDA$beta

discrimSPLSDA <- coef(outSPLSDA)
```

```
## Count of non zero coefficients

non_zero_coef <- sum(out$discrim[,1] != 0)

non_zero_coefFL <- sum(outFL$discrim[,1] != 0)

non_zero_coefSDA <- sum(outSDA$beta[,1] != 0)

non_zero_coefSPLSDA <- sum(coef(outSPLSDA)[,1] != 0)


## Predictions

pred <- predict(out, xte=test_X)

predFL <- predict(outFL, xte=test_X)

predSDA <- ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class1", 1,ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class2", 2, 0))

predSPLSDA <- predict(outSPLSDA, newx=test_X, type="fit")


## Test Error Rates

error <- sum(test_Y != pred$ypred)/nrow(test_X)

errorFL <- sum(test_Y != predFL$ypred)/nrow(test_X)

errorSDA <- sum(test_Y != predSDA)/nrow(test_X)

errorSPLSDA <- sum(test_Y != as.matrix(sapply(predSPLSDA, function(x)
as.numeric(as.character(x)))))/nrow(test_X)


output <- (list(error=error, discrim=discrim, non_zero_coef=non_zero_coef,

errorFL=errorFL, discrimFL=discrimFL, non_zero_coefFL=non_zero_coefFL,

errorSPLSDA=errorSPLSDA, discrimSPLSDA=discrimSPLSDA,
non_zero_coefSPLSDA=non_zero_coefSPLSDA, errorSDA=errorSDA,
non_zero_coefSDA=non_zero_coefSDA)   )

return(output)

}

simulation()
```

```
result[k, 1] <- k

result[k, 2] <- simulation()$error

result[k, 3] <- simulation()$non_zero_coef

result[k, 4] <- simulation()$errorFL

result[k, 5] <- simulation()$non_zero_coefFL

result[k, 6] <- simulation()$errorSPLSDA

result[k, 7] <- simulation()$non_zero_coefSPLSDA

result[k, 8] <- simulation()$errorSDA

result[k, 9] <- simulation()$non_zero_coefSDA


## matrix of coefficients

nzerobetas[[k]] <- simulation()$discrim

nzerobetasFL[[k]] <- simulation()$discrimFL

nzerobetasSPLSDA[[k]] <- simulation()$discrimSPLSDA

nzerobetasSDA[[k]] <- simulation()$discrimSDA


coefL1 <- t(matrix(unlist(nzerobetas), ncol = p, byrow = FALSE))

coefFL <- t(matrix(unlist( nzerobetasFL), ncol = p, byrow = FALSE))

coefSPLSDA <- t(matrix(unlist(nzerobetasSPLSDA), ncol = p, byrow = FALSE))


obj <- (list(result=result, coefL1=coefL1, coefFL=coefFL,
coefSPLSDA=coefSPLSDA))

end.time <- Sys.time()

cat("Iteration", k, "out of", reps, "is completed.", "Time taken is",
(end.time - start.time), "minutes.","\n")

 beep(2)

if (k == reps) cat("Done!\n")
```

```
}

return(obj)

}

simu3 <- Simulation3()


###### Simulation 4: unequal means, equal covariance and uncorrelated
features.

Simulation4 <- function(){

#### Simlation4 performs sparse classification in high-dimensional data
using the following techniques

# 1. PenalizedLDA L1

# 2. PenalizedLDA FL

# 3. SDA

# 4. SPLSDA

reps = 100 # Number of simulation repatitions

result <- matrix(nrow = reps, ncol = 9)

nzerobetas <- list()

nzerobetasFL <- list()

nzerobetasSDA <- list()

nzerobetasSPLSDA <- list()

colnames(result) <- c("Repetition", "Error.L1", "NonZeroFeatures.L1",
"Error.FL", "NonZeroFeatures.FL", "Error.SPLSDA", "NonZeroFeatures.SPLSDA",

"Error.SDA", "NonZeroFeatures.SDA")

#### Simulation study for classification in high-dimensional

for (k in 1:reps) {

start.time <- Sys.time()

simulation <- function()

{

library(MASS)
```

```
library(penalizedLDA)
library(sparseLDA)
library(rgl)
library(spls)
library(beepr)


p = 500  # p = number of features
## Training data
mu1 <- matrix(c(rep(0.3,100), rep(0, 400)))
mu2 <- matrix(c(rep(0,100), rep(0.3, 100), rep(0, 300)))


sig <- matrix(0.7 ,p ,p)
diag(sig) <- 1


n1=n2=50
Class1 <- mvrnorm(n1, mu1, sig)
Class2 <- mvrnorm(n2, mu2, sig)
train_X <- rbind(Class1, Class2)
train_Y <- matrix(c(rep(1, n1), rep(2, n2)))


Y1 <- matrix(c(rep(1, n1), rep(0, n2)))
Y2 <- matrix(c(rep(0, n1), rep(1, n2)))
Y_train <- cbind(Y1, Y2)
colnames(Y_train) <- c("Class1", "Class2")


## Test data
m1=m2=1000
Class.1 <- mvrnorm(m1, mu1, sig)
```

```
Class.2 <- mvrnorm(m2, mu2, sig)

test_X <- rbind(Class.1, Class.2)

test_Y <- matrix(c(rep(1, m1), rep(2, m2)))


Y.1 <- matrix(c(rep(1, m1), rep(0, m2)))

Y.2 <- matrix(c(rep(0, m1), rep(1, m2)))

X_test <- cbind(Y.1, Y.2)

colnames(X_test) <- c("Class1", "Class2")


################## PenalizedLDA ###################

### If type="standard" then we make use of lasso penalties.

cvL1 <- PenalizedLDA.cv(x=train_X, y=train_Y, type="standard",
lambdas=c(1e-4, 1e-3, 1e-2, 0.1, 1, 10), nfold=10)

out <- PenalizedLDA(x=train_X, y=train_Y, type="standard",
lambda=cvL1$bestlambda, K=1)


### If type="ordered" then we make use of lasso penalties.

cvFL <- PenalizedLDA.cv(x=train_X, y=train_Y, type="ordered", lambdas=c(1e-
4, 1e-3, 1e-2, 0.1, 1, 10), lambda2=0.4, nfold=10)

outFL <- PenalizedLDA(x=train_X, y=train_Y, type="ordered",
lambda=cvFL$bestlambda, K=1, lambda2=0.4)


################### SPLSDA ######################

### Select tuning parameter K (number of latent components)

cvSPLSDA <- cv.spls(x=train_X, y=train_Y, fold=10, K=1:10,
eta=seq(0.1,0.9,0.1), kappa=0.5, plot.it=FALSE)

outSPLSDA <- splsda(x=train_X, y=train_Y, K=cvSPLSDA$K.opt,
eta=cvSPLSDA$eta.opt, kappa=0.5, scale.x=FALSE, classifier ='lda')


#################### SDA ########################
```

```
Xc <- normalize(train_X)

Xn <- Xc$Xc

outSDA <- sda(x=Xn, y=Y_train, lambda=0.01, stop=0.001, maxIte=10,
trace=FALSE)


## Selected features

discrim <- out$discrim

discrimFL <- outFL$discrim

discrimSDA <- outSDA$beta

discrimSPLSDA <- coef(outSPLSDA)


## Count of non zero coefficients

non_zero_coef <- sum(out$discrim[,1] != 0)

non_zero_coefFL <- sum(outFL$discrim[,1] != 0)

non_zero_coefSDA <- sum(outSDA$beta[,1] != 0)

non_zero_coefSPLSDA <- sum(coef(outSPLSDA)[,1] != 0)


## Predictions

pred <- predict(out, xte=test_X)

predFL <- predict(outFL, xte=test_X)

predSDA <- ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class1", 1,ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class2", 2, 0))

predSPLSDA <- predict(outSPLSDA, newx=test_X, type="fit")


## Test Error Rates

error <- sum(test_Y != pred$ypred)/nrow(test_X)

errorFL <- sum(test_Y != predFL$ypred)/nrow(test_X)

errorSDA <- sum(test_Y != predSDA)/nrow(test_X)
```

```
errorSPLSDA <- sum(test_Y != as.matrix(sapply(predSPLSDA, function(x)
as.numeric(as.character(x)))))/nrow(test_X)


output <- (list(error=error, discrim=discrim, non_zero_coef=non_zero_coef,

errorFL=errorFL, discrimFL=discrimFL, non_zero_coefFL=non_zero_coefFL,

errorSPLSDA=errorSPLSDA, discrimSPLSDA=discrimSPLSDA,
non_zero_coefSPLSDA=non_zero_coefSPLSDA, errorSDA=errorSDA,
non_zero_coefSDA=non_zero_coefSDA)   )

return(output)

}

simulation()


result[k, 1] <- k

result[k, 2] <- simulation()$error

result[k, 3] <- simulation()$non_zero_coef

result[k, 4] <- simulation()$errorFL

result[k, 5] <- simulation()$non_zero_coefFL

result[k, 6] <- simulation()$errorSPLSDA

result[k, 7] <- simulation()$non_zero_coefSPLSDA

result[k, 8] <- simulation()$errorSDA

result[k, 9] <- simulation()$non_zero_coefSDA


## matrix of coefficients

nzerobetas[[k]] <- simulation()$discrim

nzerobetasFL[[k]] <- simulation()$discrimFL

nzerobetasSPLSDA[[k]] <- simulation()$discrimSPLSDA

nzerobetasSDA[[k]] <- simulation()$discrimSDA


coefL1 <- t(matrix(unlist(nzerobetas), ncol = p, byrow = FALSE))
```

```
coefFL <- t(matrix(unlist( nzerobetasFL), ncol = p, byrow = FALSE))

coefSPLSDA <- t(matrix(unlist(nzerobetasSPLSDA), ncol = p, byrow = FALSE))


obj <- (list(result=result, coefL1=coefL1, coefFL=coefFL,
coefSPLSDA=coefSPLSDA))

end.time <- Sys.time()

cat("Iteration", k, "out of", reps, "is completed.", "Time taken is",
(end.time - start.time), "minutes.","\n")

 beep(2)

if (k == reps) cat("Done!\n")

}

return(obj)

}

simu4 <- Simulation4()


###### Simulation 5: unequal means, equal covariance and uncorrelated
features.

Simulation5 <- function(){

#### Simlation5 performs sparse classification in high-dimensional data
using the following techniques

# 1. PenalizedLDA L1

# 2. PenalizedLDA FL

# 3. SDA

# 4. SPLSDA

reps = 100 # Number of simulation repatitions

result <- matrix(nrow = reps, ncol = 9)

nzerobetas <- list()

nzerobetasFL <- list()

nzerobetasSDA <- list()
```

```
nzerobetasSPLSDA <- list()

colnames(result) <- c("Repetition", "Error.L1", "NonZeroFeatures.L1",
"Error.FL", "NonZeroFeatures.FL", "Error.SPLSDA", "NonZeroFeatures.SPLSDA",

"Error.SDA", "NonZeroFeatures.SDA")

#### Simulation study for classification in high-dimensional

for (k in 1:reps) {

start.time <- Sys.time()

simulation <- function()

{

library(MASS)

library(penalizedLDA)

library(sparseLDA)

library(rgl)

library(spls)

library(beepr)


p = 500  # p = number of features

## Training data

mu1 <- matrix(c(rep(0.3,100), rep(0, 400)))

mu2 <- matrix(c(rep(0,100), rep(0.3, 100), rep(0, 300)))


sig <- matrix(0.7 ,p ,p)

diag(sig) <- 1


n1=n2=50

Class1 <- mvrnorm(n1, mu1, sig)

Class2 <- mvrnorm(n2, mu2, sig)

train_X <- rbind(Class1, Class2)
```

```
train_Y <- matrix(c(rep(1, n1), rep(2, n2)))


Y1 <- matrix(c(rep(1, n1), rep(0, n2)))

Y2 <- matrix(c(rep(0, n1), rep(1, n2)))

Y_train <- cbind(Y1, Y2)

colnames(Y_train) <- c("Class1", "Class2")


## Test data

m1=m2=1000

Class.1 <- mvrnorm(m1, mu1, sig)

Class.2 <- mvrnorm(m2, mu2, sig)

test_X <- rbind(Class.1, Class.2)

test_Y <- matrix(c(rep(1, m1), rep(2, m2)))


Y.1 <- matrix(c(rep(1, m1), rep(0, m2)))

Y.2 <- matrix(c(rep(0, m1), rep(1, m2)))

X_test <- cbind(Y.1, Y.2)

colnames(X_test) <- c("Class1", "Class2")


################## PenalizedLDA ###################

### If type="standard" then we make use of lasso penalties.

cvL1 <- PenalizedLDA.cv(x=train_X, y=train_Y, type="standard",
lambdas=c(1e-4, 1e-3, 1e-2, 0.1, 1, 10), nfold=10)

out <- PenalizedLDA(x=train_X, y=train_Y, type="standard",
lambda=cvL1$bestlambda, K=1)


### If type="ordered" then we make use of lasso penalties.

cvFL <- PenalizedLDA.cv(x=train_X, y=train_Y, type="ordered", lambdas=c(1e-
4, 1e-3, 1e-2, 0.1, 1, 10), lambda2=0.4, nfold=10)
```

```
outFL <- PenalizedLDA(x=train_X, y=train_Y, type="ordered",
lambda=cvFL$bestlambda, K=1, lambda2=0.4)


################### SPLSDA ######################

### Select tuning parameter K (number of latent components)

cvSPLSDA <- cv.spls(x=train_X, y=train_Y, fold=10, K=1:10,
eta=seq(0.1,0.9,0.1), kappa=0.5, plot.it=FALSE)

outSPLSDA <- splsda(x=train_X, y=train_Y, K=cvSPLSDA$K.opt,
eta=cvSPLSDA$eta.opt, kappa=0.5, scale.x=FALSE, classifier ='lda')


#################### SDA ########################

Xc <- normalize(train_X)

Xn <- Xc$Xc

outSDA <- sda(x=Xn, y=Y_train, lambda=0.01, stop=0.001, maxIte=10,
trace=FALSE)


## Selected features

discrim <- out$discrim

discrimFL <- outFL$discrim

discrimSDA <- outSDA$beta

discrimSPLSDA <- coef(outSPLSDA)


## Count of non zero coefficients

non_zero_coef <- sum(out$discrim[,1] != 0)

non_zero_coefFL <- sum(outFL$discrim[,1] != 0)

non_zero_coefSDA <- sum(outSDA$beta[,1] != 0)

non_zero_coefSPLSDA <- sum(coef(outSPLSDA)[,1] != 0)


## Predictions
```

```
pred <- predict(out, xte=test_X)

predFL <- predict(outFL, xte=test_X)

predSDA <- ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class1", 1,ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class2", 2, 0))

predSPLSDA <- predict(outSPLSDA, newx=test_X, type="fit")


## Test Error Rates

error <- sum(test_Y != pred$ypred)/nrow(test_X)

errorFL <- sum(test_Y != predFL$ypred)/nrow(test_X)

errorSDA <- sum(test_Y != predSDA)/nrow(test_X)

errorSPLSDA <- sum(test_Y != as.matrix(sapply(predSPLSDA, function(x)
as.numeric(as.character(x)))))/nrow(test_X)


output <- (list(error=error, discrim=discrim, non_zero_coef=non_zero_coef,

errorFL=errorFL, discrimFL=discrimFL, non_zero_coefFL=non_zero_coefFL,

errorSPLSDA=errorSPLSDA, discrimSPLSDA=discrimSPLSDA,
non_zero_coefSPLSDA=non_zero_coefSPLSDA, errorSDA=errorSDA,
non_zero_coefSDA=non_zero_coefSDA)   )

return(output)

}

simulation()


result[k, 1] <- k

result[k, 2] <- simulation()$error

result[k, 3] <- simulation()$non_zero_coef

result[k, 4] <- simulation()$errorFL

result[k, 5] <- simulation()$non_zero_coefFL

result[k, 6] <- simulation()$errorSPLSDA
```

```
result[k, 7] <- simulation()$non_zero_coefSPLSDA

result[k, 8] <- simulation()$errorSDA

result[k, 9] <- simulation()$non_zero_coefSDA


## matrix of coefficients

nzerobetas[[k]] <- simulation()$discrim

nzerobetasFL[[k]] <- simulation()$discrimFL

nzerobetasSPLSDA[[k]] <- simulation()$discrimSPLSDA

nzerobetasSDA[[k]] <- simulation()$discrimSDA


coefL1 <- t(matrix(unlist(nzerobetas), ncol = p, byrow = FALSE))

coefFL <- t(matrix(unlist( nzerobetasFL), ncol = p, byrow = FALSE))

coefSPLSDA <- t(matrix(unlist(nzerobetasSPLSDA), ncol = p, byrow = FALSE))


obj <- (list(result=result, coefL1=coefL1, coefFL=coefFL,
coefSPLSDA=coefSPLSDA))

end.time <- Sys.time()

cat("Iteration", k, "out of", reps, "is completed.", "Time taken is",
(end.time - start.time), "minutes.","\n")

 beep(2)

if (k == reps) cat("Done!\n")

}

return(obj)

}

simu5 <- Simulation5()


## Mean and Standard Error function for Error rates and number of features

simu1_mean = round(apply((simu1$result[,-1]) ,2, mean), 2)
```

```
simu1_mean

simu2_mean = round(apply((simu2$result[,-1]) ,2, mean), 2)

simu2_mean

simu3_mean = round(apply((simu3$result[,-1]) ,2, mean), 2)

simu3_mean

simu4_mean = round(apply((simu4$result[,-1]) ,2, mean), 2)

simu4_mean

simu5_mean = round(apply((simu5$result[,-1]) ,2, mean), 2)

simu5_mean


simu1_sd = round(apply((simu1$result[,-1]) ,2, sd), 2)

simu1_sd

simu2_sd = round(apply((simu2$result[,-1]) ,2, sd), 2)

simu2_sd

simu3_sd = round(apply((simu3$result[,-1]) ,2, sd), 2)

simu3_sd

simu4_sd = round(apply((simu4$result[,-1]) ,2, sd), 2)

simu4_sd

simu5_sd = round(apply((simu5$result[,-1]) ,2, sd), 2)

simu5_sd

################# Box Plots: Test Errors ####################

par(mfrow=c(1,2))

dat1 <- cbind(simu1$result[,2], simu1$result[,4], simu1$result[,6],
simu1$result[,8])

boxplot(dat1, main="Simulation 1: Boxplots of overall test error rates",
ylab="Test error rates", xlab="Classification technique",

col = c("red","sienna","palevioletred1","royalblue2"), names =
c("penLDA_L1","penLDA_FL","SPLSDA","SDA"))
```

```
dat2 <- cbind(simu2$result[,2], simu2$result[,4], simu2$result[,6],
simu2$result[,8])

boxplot(dat2, main="Simulation 2: Boxplots of overall test error rates",
ylab="Test error rates", xlab="Classification technique",

col = c("red","sienna","palevioletred1","royalblue2"), names =
c("penLDA_L1","penLDA_FL","SPLSDA","SDA"))


par(mfrow=c(1,2))

dat3 <- cbind(simu3$result[,2], simu3$result[,4], simu3$result[,6],
simu3$result[,8])

boxplot(dat3, main="Simulation 3: Boxplots of overall test error rates",
ylab="Test error rates", xlab="Classification technique",

col = c("red","sienna","palevioletred1","royalblue2"), names =
c("penLDA_L1","penLDA_FL","SPLSDA","SDA"))


dat4 <- cbind(simu4$result[,2], simu4$result[,4], simu4$result[,6],
simu4$result[,8])

boxplot(dat4, main="Simulation 4: Boxplots of overall test error rates",
ylab="Test error rates", xlab="Classification technique",

col = c("red","sienna","palevioletred1","royalblue2"), names =
c("penLDA_L1","penLDA_FL","SPLSDA","SDA"))


par(mfrow=c(1,2))

dat5 <- cbind(simu5$result[,2], simu5$result[,4], simu5$result[,6],
simu5$result[,8])

boxplot(dat5, main="Simulation 5: Boxplots of overall test error rates",
ylab="Test error rates", xlab="Classification technique",

col = c("red","sienna","palevioletred1","royalblue2"), names =
c("penLDA_L1","penLDA_FL","SPLSDA","SDA"))
```

B. R function for Empirical Studies.

```
study_HD <- function(data, reps)
{
#Reps = Number of repetitions.
#Data = Microarray gene expression data with a binary outcome.

### Load libraries
library(MASS)
library(penalizedLDA)
library(sparseLDA)
library(rgl)
library(spls)
library(randomForest)
library(beepr)

p <- (ncol(data)-1)

result <- matrix(nrow = reps, ncol = 11)
result2 <- matrix(nrow = reps, ncol = 11)
nzerobetasL1 <- list()
nzerobetasFL <- list()
nzerobetasSPLSDA <- list()
nzerobetasSDA <- list()
nzerobetasRF <- matrix(nrow = p, ncol = reps)

colnames(result) <- c("Repetition", "Error.L1", "NonZeroFeatures.L1",
"Error.FL", "NonZeroFeatures.FL","Error.SPLSDA", "NonZeroFeatures.SPLSDA",
"Error.SDA", "NonZeroFeatures.SDA", "Error.RF", "NonZeroFeatures.RF")

colnames(result2) <- c("Repetition", "spec.L1", "sens.L1", "spec.FL",
"sens.FL", "spec.SPLSDA", "sens.SPLSDA", "spec.SDA", "sens.SDA", "spec.RF",
"sens.RF")

      ### the repetition loop starts here
for (k in 1:reps)
```

```
{
start.time <- Sys.time()
simulation <- function(Dataset=data)
{
#### Emperical study for classification in high-dimension ####
#### DATA SPLITTING ###
Yvec <- Dataset[,1]
n1 <- sum(Yvec == 1)
n2 <- sum(Yvec == 2)
n12 <- n1 + n2
ni <- c(n1, n2)
ngroup <- length(ni)

## Training fraction
perc <- 0.7
pos <- c(0, cumsum(ni))
proportion <- floor(perc * ni)
## Sample training and test cases from the two groups ##
Indexes <- NULL
TrainInd <- NULL
for (i in 1:ngroup)
{
Indexes[[i]] <- sample(((pos[i] + 1):(pos[i + 1])), size = proportion[[i]],
replace = FALSE)
}
TrainInd <- c(Indexes[[1]], Indexes[[2]])
training <- Dataset[TrainInd, ]
testing <- Dataset[-TrainInd, ]

train_X <- training[,-1]
train_Y <- as.matrix(training[,1])
Y1 <- matrix(c(rep(1, length(Indexes[[1]])), rep(0, length(Indexes[[2]]))))
Y2 <- matrix(c(rep(0, length(Indexes[[1]])), rep(2, length(Indexes[[2]]))))
Y_train <- cbind(Y1, Y2)
```

```
colnames(Y_train) <- c("Class1", "Class2")

test_X <- testing[,-1]

test_Y <- as.matrix(testing[,1])


################## PenalizedLDA ###################

### If type="standard" then we make use of lasso penalties.

cvL1 <- PenalizedLDA.cv(x=train_X, y=train_Y, type="standard",
lambdas=c(1e-4, 1e-2, 0.1, 1), nfold=5)

outL1 <- PenalizedLDA(x=train_X, y=train_Y, type="standard",
lambda=cvL1$bestlambda, K=1)

### If type="ordered" then we make use of lasso penalties.

cvFL <- PenalizedLDA.cv(x=train_X, y=train_Y, type="ordered", lambdas=c(1e-
4, 1e-2, 0.1, 1), lambda2=0.4, nfold=5)

outFL <- PenalizedLDA(x=train_X, y=train_Y, type="ordered",
lambda=cvFL$bestlambda, K=1, lambda2=0.4)


################### SPLSDA ######################

#cvSPLSDA <- cv.spls(x=train_X, y=train_Y, fold=5, K=c(1:10),
eta=seq(0.1,0.9,0.1), kappa=0.5, plot.it=FALSE)

#outSPLSDA <- splsda(x=train_X, y=train_Y, K=cvSPLSDA$K.opt,
eta=cvSPLSDA$eta.opt, kappa=0.5, scale.x=FALSE, classifier='lda')

outSPLSDA <- splsda(x=train_X, y=train_Y, K=4, eta=0.8, kappa=0.5,
scale.x=FALSE, classifier='lda')


#################### SDA ########################

Xc <- normalize(train_X)

Xn <- Xc$Xc

outSDA <- sda(x=Xn, y=Y_train, lambda=0.01, stop=0.001, maxIte=10,
trace=FALSE)


################ Random Forest ##################

outRF <- randomForest(x=train_X, y=as.factor(train_Y), importance=TRUE,
ntree=500)

## Selected features

discrimL1 <- outL1$discrim

discrimFL <- outFL$discrim

discrimSPLSDA <- coef(outSPLSDA)

discrimRF <- as.matrix(outRF$importance[,1])
```

```
index <- as.matrix(rep(1:p))
colnames(index) <- c("Index")
SDA.mat <- cbind(outSDA$varIndex, outSDA$beta)
colnames(SDA.mat) <- c("Index", "discrimSDA")
mat <- merge(index,  SDA.mat, by=c("Index"), all=TRUE)
mat[is.na(mat)] <- 0
discrimSDA <- mat[,2]

      ## Count of non zero coefficients
non_zero_coefL1 <- sum(outL1$discrim[,1] != 0)
non_zero_coefFL <- sum(outFL$discrim[,1] != 0)
non_zero_coefSPLSDA <- sum(coef(outSPLSDA)[,1] != 0)
non_zero_coefSDA <- sum(outSDA$beta[,1] != 0)
non_zero_coefRF <- sum(outRF$importance[,1] != 0)

      ## Predictions
predL1 <- predict(outL1, xte=test_X)
predFL <- predict(outFL, xte=test_X)
predSPLSDA <- predict(outSPLSDA, newx=test_X, type="fit")
predSDA <- ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class1", 1, ifelse(predict(outSDA, normalizetest(test_X, Xc))$class ==
"Class2", 2, 0))
predRF <- predict(outRF, newdata=test_X, type="response")

      ## Test Error
errorL1 <- sum(test_Y != predL1$ypred)
errorFL <- sum(test_Y != predFL$ypred)
errorSPLSDA <- sum(test_Y != as.matrix(sapply(predSPLSDA, function(x)
as.numeric(as.character(x)))))
errorSDA <- sum(test_Y != predSDA)
errorRF <- sum(test_Y != predRF)

      ##Confusion Matrix
confmatL1 <- table( predL1$ypred, test_Y)
```

```
confmatFL <- table(predFL$ypred, test_Y)

confmatSPLSDA <- table(as.matrix(as.numeric(predSPLSDA)), test_Y)

confmatSDA <- table(as.matrix(predSDA), test_Y)

confmatRF <- table(as.matrix(as.numeric(predRF)), test_Y)


      ## Specificity

specificityL1 <- round(confmatL1[2,2]/(confmatL1[2,2]+confmatL1[1,2]),2)

specificityFL <- round(confmatFL[2,2]/(confmatFL[2,2]+confmatFL[1,2]),2)

specificitySPLSDA <-
round(confmatSPLSDA[2,2]/(confmatSPLSDA[2,2]+confmatSPLSDA[1,2]),2)

specificitySDA <-
round(confmatSDA[2,2]/(confmatSDA[2,2]+confmatSDA[1,2]),2)

specificityRF <- round(confmatRF[2,2]/(confmatRF[2,2]+confmatRF[1,2]),2)


            ## Sensitivity

sensitivityL1 <- round(confmatL1[1,1]/(confmatL1[1,1]+confmatL1[2,1]),2)

sensitivityFL <- round(confmatFL[1,1]/(confmatFL[1,1]+confmatFL[2,1]),2)

sensitivitySPLSDA <-
round(confmatSPLSDA[1,1]/(confmatSPLSDA[1,1]+confmatSPLSDA[2,1]),2)

sensitivitySDA <-
round(confmatSDA[1,1]/(confmatSDA[1,1]+confmatSDA[2,1]),2)

sensitivityRF <- round(confmatRF[1,1]/(confmatRF[1,1]+confmatRF[2,1]),2)


output <- (list(errorL1=errorL1, discrimL1=discrimL1,
non_zero_coefL1=non_zero_coefL1, specificityL1=specificityL1,
sensitivityL1=sensitivityL1,errorFL=errorFL, discrimFL=discrimFL,
non_zero_coefFL=non_zero_coefFL, specificityFL=specificityFL,
sensitivityFL=sensitivityFL, errorSPLSDA=errorSPLSDA,
discrimSPLSDA=discrimSPLSDA, non_zero_coefSPLSDA=non_zero_coefSPLSDA,

specificitySPLSDA=specificitySPLSDA, sensitivitySPLSDA=sensitivitySPLSDA,

errorSDA=errorSDA, discrimSDA=discrimSDA,
non_zero_coefSDA=non_zero_coefSDA,

specificitySDA=specificitySDA, sensitivitySDA=sensitivitySDA,

errorRF=errorRF, discrimRF=discrimRF, non_zero_coefRF=non_zero_coefRF,

specificityRF=specificityRF, sensitivityRF=sensitivityRF

)   )

return(output)

}

simulation()
```

```
        ## Matrix of error rates and number of coefficients
result[k, 1] <- k
result[k, 2] <- simulation()$errorL1
result[k, 3] <- simulation()$non_zero_coefL1
result[k, 4] <- simulation()$errorFL
result[k, 5] <- simulation()$non_zero_coefFL
result[k, 6] <- simulation()$errorSPLSDA
result[k, 7] <- simulation()$non_zero_coefSPLSDA
result[k, 8] <- simulation()$errorSDA
result[k, 9] <- simulation()$non_zero_coefSDA
result[k, 10] <- simulation()$errorRF
result[k, 11] <- simulation()$non_zero_coefRF

      ## Matrix of sensitivity and specificity
result2[k, 1] <- k
result2[k, 2] <- simulation()$specificityL1
result2[k, 3] <- simulation()$sensitivityL1
result2[k, 4] <- simulation()$specificityFL
result2[k, 5] <- simulation()$sensitivityFL
result2[k, 6] <- simulation()$specificitySPLSDA
result2[k, 7] <- simulation()$sensitivitySPLSDA
result2[k, 8] <- simulation()$specificitySDA
result2[k, 9] <- simulation()$sensitivitySDA
result2[k, 10] <- simulation()$specificityRF
result2[k, 11] <- simulation()$sensitivityRF

        ## matrix of coefficients
nzerobetasL1[[k]] <- simulation()$discrimL1
nzerobetasFL[[k]] <- simulation()$discrimFL
nzerobetasSPLSDA[[k]] <- simulation()$discrimSPLSDA
nzerobetasSDA[[k]] <- simulation()$discrimSDA
nzerobetasRF[,k] <- simulation()$discrimRF
```

```
coefL1 <- t(matrix(unlist(nzerobetasL1), ncol = p, byrow = FALSE))

coefFL <- t(matrix(unlist( nzerobetasFL), ncol = p, byrow = FALSE))

coefSPLSDA <- t(matrix(unlist(nzerobetasSPLSDA), ncol = p, byrow = FALSE))

coefSDA <- t(matrix(unlist(nzerobetasSDA), ncol = p, byrow = FALSE))


obj <- (list(result=result, result2=result2, coefL1=coefL1, coefFL=coefFL,
coefSPLSDA=coefSPLSDA, coefSDA=coefSDA, coefRF=nzerobetasRF))

end.time <- Sys.time()

cat("Iteration", k, "out of", reps,"is completed." ,"Time taken is",
(end.time - start.time), "minutes.","\n")

beep(2)

if (k == reps) cat("Done!\n")

}

return(obj)

}


library(rda)

data(colon)

Colon <- as.matrix(cbind(as.matrix(colon.y), scale(colon.x)))

study1 <- study_HD(data=Colon, reps=50)


library(varbvs)

data(leukemia)

Leukemia <- cbind(leukemia$y+1, scale(leukemia$x))

study2 <- study_HD(data=Leukemia, reps=50)


############ Mean and Standard Error function for the Error rates, number
of features, sensitivity and specificity #########

study1_mean = round(apply((study1$result[,-1]/100) ,2, mean), 4)

study1_mean

study2_mean = round(apply((study2$result[,-1]/100) ,2, mean), 4)

study2_mean


study1_sd = round(apply((study1$result[,-1]/100) ,2, sd), 2)

study1_sd
```

```
study2_sd = round(apply(study2$result[,-1]/100 ,2, sd), 2)

study2_sd


study1_mean2 = round(apply(study1$result2[,-1] ,2, mean), 2)

study1_mean2

study2_mean2 = round(apply(study2$result2[,-1] ,2, mean), 2)

study2_mean2


study1_sd2 = round(apply(study1$result2[,-1] ,2, sd), 2)

study1_sd2

study2_sd2 = round(apply(study2$result2[,-1] ,2, sd), 2)

study2_sd2


################ Box Plots: Test Errors ###################

par(mfrow=c(1,2))

res1 <- cbind(study1$result[,2], study1$result[,4], study1$result[,6],
study1$result[,8], study1$result[,10])

boxplot(res1, main="Colon cancer dataset: Boxplots of overall test error
rates", ylab="Test error rate", xlab="Classification technique",

col = c("red", "sienna", "palevioletred1", "royalblue2", "orange"), names =
c("penalizedLDA_L1", "penalizedLDA_FL", "SPLSDA", "SDA", "RF"))


res2 <- cbind(study2$result[,2], study2$result[,4], study2$result[,6],
study2$result[,8], study2$result[,10])

boxplot(res2, main="Leukemia cancer dataset: Boxplots of overall test error
rates", ylab="Test error rate", xlab="Classification technique",

col = c("red", "sienna", "palevioletred1", "royalblue2", "orange"), names =
c("penalizedLDA_L1", "penalizedLDA_FL", "SPLSDA", "SDA", "RF"))
```